COORDINATING DECENTRALIZED LEARNING AND CONFLICT RESOLUTION ACROSS AGENT BOUNDARIES

by

Shanjun Cheng

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2012

Approved by:

_____
Dr. Anita Raja

_____
Dr. Jing Xiao

_____
Dr. Mirsad Hadzikadic

_____
Dr. Jiang (Linda) Xie

_____
Dr. Mary Maureen Brown

ABSTRACT

SHANJUN CHENG.Coordinating decentralized learning and conflict resolution across agent boundaries. (Under the direction of DR. ANITA RAJA)

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 because of scalability, partial information accessibility and complex interaction of agents. It is a challenge for agents to learn good policies, when they need to plan and coordinate in uncertain, dynamic environments, especially when they have large state spaces. It is also critical for agents operating in a multiagent system (MAS) to resolve conflicts among the learned policies of different agents, since such conflicts may have detrimental influence on the overall performance.

The focus of this research is to use a reinforcement learning based local optimization algorithm within each agent to learn multiagent 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. The research takes an adaptive approach to resolving conflicts that can arise between locally optimal agent policies. First an algorithm that uses heuristic rules to locally resolve simple conflicts is presented. When the environment is more dynamic and uncertain, a mediator-based mechanism to resolve more complicated conflicts and selectively expand the agents' state space during the learning process is harnessed. For scenarios where mediator-based mechanisms with partially global views are ineffective, a more rigorous approach for global conflict resolution that synthesizes multiagent reinforcement learning (MARL) and distributed constraint optimization (DCOP) is developed. These mechanisms are evaluated in the context of a multiagent tornado tracking application called NetRads. Empirical results show that these mechanisms significantly improve the performance of the tornado tracking network for a variety

of weather scenarios.

The major contributions of this work are: a state of the art decentralized learning approach that supports agent interactions and reorganizes the underlying network when needed; the use of abstract classes of scenarios/states/actions that efficiently manages the exploration of the search space; novel conflict resolution algorithms of increasing complexity that use heuristic rules, sophisticated automated negotiation mechanisms and distributed constraint optimization methods respectively; and finally, a rigorous study of the interplay between two popular theories used to solve multiagent problems, namely decentralized Markov decision processes and distributed constraint optimization.

ACKNOWLEDGMENTS

I am deeply grateful to my advisor Dr. Anita Raja, who has given her time and talent in the ways that simply can not be adequately acknowledged. Dr. Raja has provided constant, immeasurable intellectual guidance and inspiration. I benefited greatly from her energy and enthusiasm throughout my five-year study terms. To me, she served all the roles of teacher, mentor and friend. She has cared about my research progress, personal life as well as my family. It is my honor to be one of her students, and I am very thankful for the opportunity to learn from Dr. Raja.

I also especially want to thank my other research collaborators Dr. Victor Lesser, Dr. Jiang (Linda) Xie and Dr. Ivan Howitt. They have served to help me on research development greatly. I am grateful to Dr. Lesser for his insightful and rigorous comments both on my work and on this dissertation. I am thankful to Dr. Xie for her hard work on the revisions of my papers. I would also like to thank Dr. Howitt for his motivating discussions.

I would also like to thank my other dissertation committee members Dr. Jing Xiao, Dr. Mirsad Hadzikadic and Dr. Mary Maureen Brown. I am thankful to their insightful comments on my dissertation and probing and stimulating questions during my defense.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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. For example, in sensor networks, because of limited communication bandwidth, the control authority is naturally distributed among sensors, which work together to achieve some common goal (e.g., tracking weather phenomena). In electricity grids, electricity distribution management is decentralized among power stations, which coordinate their power control configurations in order to satisfy variable demands from all customers and minimize losses. These agents operate in an iterative three-step closed loop [Russell and Norvig, 2006]: 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 the three-step closed loop: deliberative and meta-level control. The lower level is deliberative control, which involves the agent making decisions about what local problem solving to perform in the current context (also called domain actions) and how to coordinate with other agents to complete tasks requiring joint effort. These deliberations may have to be done in the face of limited resources, uncertainty about action outcomes and in real-time. Tasks in these environments can be generated at any time by the environment or other agents and generally have deadlines where completion after the deadline could lead to lower or no utility.

At the higher level is meta-level control, which involves the agent making decisions about whether to deliberate, how many resources to dedicate to this deliberation and what specific deliberative control to perform in the current context. In practice, meta-level control can be viewed as the process of deciding how to interleave domain and deliberative control actions such that tasks are achieved within their deadlines. It also involves allocating required amount of processor and other resources to these actions at the appropriate times. The decision-making criterion for this process is to maximize the overall utility as measured by the utility obtained by completing individual tasks. For example, suppose the current time is 10 and an agent is in the midst of executing a set of high quality [1] tasks with a deadline to complete the task at time 25. At time 15 the agent receives a new medium quality task T_{new} with expected duration of 10 and a deadline of 40. The sensible meta control decision would be for the agent to delay deliberating about how to accomplish task T_{new} in the context of ongoing activities until the existing task set has completed execution (time 25). This would guarantee that the existing task set completes within its deadline, and quality can still be gained by processing T_{new} by time 40. The meta-level control decision process should be designed to be computationally inexpensive, thus obviating the need for meta-meta-level control.

Meta-level control also involves making choices from alternative deliberative action sequences including choosing among various alternatives for scheduling/planning [2]; choosing between scheduling/planning and coordination; and, allocating extra time for learning activities, etc. Consider the following example: suppose the current time is 6 and an agent has two tasks: T_x, a high quality task with expected duration of 10 and deadline 30; and T_y, a low quality task with expected duration 6 and deadline

[1] Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Thus, different applications have different notions of what corresponds to model quality.

[2] Planning refers to the process of generation of a sequence of activities that satisfy the requirements for a task and scheduling is allocating the appropriate resources for the plan to be accomplished.

30. The meta-control decision could be to spend 5 time units doing a detailed high-quality deliberation about T_x to find a good plan for the high quality task and two units doing a quick and dirty deliberation to generate a plan for T_y (the lower quality task). The remaining time not used by deliberative activities will be allocated to successfully executing both tasks.

In a cooperative MAS, each agent selects actions individually, but it is the resulting joint action that produces the outcome. It is critical to ensure that the individual decisions of the agents result in (near-)optimal decisions for the group as a whole. A central challenge in cooperative MAS research is to design distributed decision policies for agents to coordinate their actions in order to efficiently achieve their common goal. A common offline approach is to build a model (e.g., decentralized Markov decision process [Bernstein et al., 2000]) for distributed decision problems in a cooperative MAS and then compute coordination policies for agents from the model. However, this approach is usually infeasible for large-scale complex MAS applications, which involve tens to thousands of agents with limited communication bandwidth and partial views of the whole system. First, it is time-costly or even not possible to obtain an accurate model of practical MAS applications [Zhang et al., 2009]. This is especially true for application operating in open environments where the environmental characteristics are not known a priori and may evolve over time. Second, even when we have such models, the computation for optimal policies for agents is usually intractable.

Multiagent reinforcement learning (MARL) [Crites and Barto, 1995] is a common approach for solving multiagent decision making problems. It allows agents to dynamically adapt to changes in the environment, while requiring minimum domain knowledge. There are three key challenges for MARL. One is the non-stationary environment of MARL where agents are concurrently learning and adapting to one another. Another key challenge of MARL is its scalability to realistic problems, which

is an even greater cause for concern in large-scale multiagent settings. The third challenge of MARL is its exposure to conflicts among agents' actions, especially in agent networks with high inter-agent dependencies.

The dissertation of this work is to design and implement MARL techniques that support multiagent meta-level decision making. It involves modeling the meta-level problem as a decentralized coordination problem; constructing classes of scenarios where instances within a class have similar features; using the interplay between individual agent reinforcement learning and optimization techniques to create a multiagent policy where the content and timing of deliberations are choreographed carefully and include branches to account for what could happen as deliberation (and execution) plays out.

1.1 Motivation

1.1.1 Research Objective

As embedded systems consisting of collaborating agents capable of interaction become ubiquitous, they must be able to adapt to the dynamic, uncertain characteristics of an open environment. This adaptation needs to be distributed and based on the priority of tasks, availability of resources, problem-solving state of other agents, and availability of alternative ways of satisfying these and future tasks. The research objective of this dissertation is to design and develop bounded-rational agents equipped with reasoning algorithms for multiagent meta-level control (MMLC). The meta-level capabilities will allow the agents to individually 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. Consider for instance a scenario where two agents A_1 and A_2 are negotiating about when A_1 can complete method M_1 that enables A_2's method M_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 commit-

ment requests; this process repeats until A_1 and A_2 arrive at a mutually acceptable commitment. The MMLC 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.

To the best of my knowledge, Raja and Lesser [Raja and Lesser, 2003] [Raja and Lesser, 2007] were the first to explore meta-level control in complex agent-based settings where agents with a complex architecture could reason about alternative methods for computation, including computations that handled simple negotiation between two agents. The research builds on results on their work and more recent related work to open a new vein of inquiry by addressing issues of scalability, partial information and complex interactions across agent boundaries in real-world domains. My intent for this dissertation is to design and develop a domain-independent framework for MMLC including designing and implementing domain-independent versions of the offline multiagent learning and the online global optimization algorithms. The key idea is to formulate the MMLC problem as a global optimization problem with offline learning providing the cost-to-go/reward function as input allowing agents to adapt in knowledge-poor, partially observable environments. This will involve constructing a restricted class of decentralized Markov-Decision Processes (DEC-MDPs) with factored states that has the ability to communicate and model interactions so that decisions made in one agent's meta-level DEC-MDP need to be coordinated with the meta-level DEC-MDPs of other agents. This investigation will develop efficient decision-making techniques to guide problem solving in complex agents when they are faced with uncertainty, resource bounds and significant amount of interaction with other agents.

By formalizing meta-level control, this work will significantly impact multiagent systems research in that one can identify domains and scenarios where meta-level control would be advantageous to overall multiagent performance. This work will

also impact other fields that deal with uncertainty and non-stationarity including robotic path-planning. The approach is innovative in its three research goals:

- To create a meta-level framework for multiagent systems that supports coordinating decentralized Markov Decision Processes.
- To formulate the multiagent meta-level control problem as a global optimization problem that bootstraps from individual agent learning and vice-versa.
- To show the importance of meta-level control for a significant and fielded multiagent application.

1.1.2 Why is the Problem Difficult?

In the multiagent context, meta-level control decisions at different agents need to be coordinated. These agents may have multiple high-level goals from which to choose, but if two or more agents need to coordinate their actions, the agents' meta-control components must be on the same page. That is, the agents must reason about the same problem and may need to be at the same stage of the problem-solving process (e.g., if one agent decides to devote little time to communication/negotiation before moving to other deliberative decisions while another agent sets aside a large portion of deliberation time for negotiation, the latter agent would waste time trying to negotiate with an unwilling partner). Thus if an agent changes the problem-solving context it is focusing on, it must notify other agents with which it may interact. This suggests that the meta-control component of each agent should have a multiagent policy, where the progression of what deliberations agents do, and when, is choreographed carefully and includes branches to account for what could happen as deliberation (and execution) plays out [Alexander et al., 2007] [Cox and Raja, 2008].

Determining the multiagent policy is a complicated problem since it is not expected to be simply the union of all single-agent meta-control policies. In order to set up the negotiation in the example scenario discussed in the Introduction, the meta-level control should establish when negotiation results would be available. This involves

defining important parameters of the negotiation including the earliest time the target method will be enabled. Two agents with different views of meta-control policy for negotiation need to be reconciled in order to set up the earliest starting time parameter used in the negotiation process.

A second multiagent meta-level control research issue involves exploring how to dynamically split the agent network into sub-networks that are coordinated but not necessarily the same. Coordinated meta-level control decisions do not mean that meta-level control has to be the same in all parts of the network; instead, it involves finding consistent sets for different parts of the network. Multiagent meta-control suggests the need for some kind of meta-level message passing. There are important tradeoffs between the amount of communication (both size and number of messages) and resulting overhead, and the usefulness of such communication. Agents must determine what kind of information is contained in a meta-level message. In some situations, it may be enough for the agent to simply let others know that it is thinking about context X; in other cases, such as when agents are more tightly coupled, an agent may need to communicate some partial results of its current thinking as well. Agents must also reason about how to handle meta-control messages from others and coordinate when these messages should be received and handled.

In a MAS that operates in complex and evolving environments, there is a need for agents to adapt their local activities and organization relationships to the evolving characteristics of the uncertain environment. In this dissertation, we focus on decentralized systems, where an agent has only a partial view of the system, that is, an agent does not have full observability of the state of all other agents in the system. The model of decentralized partially observable Markov decision processes (DEC-POMDP) [Bernstein et al., 2002] generalizes such distributed problems. In a DEC-POMDP, all agents share the same reward function, which is called a global reward function. However, in many large-scale decentralized systems (e.g., network

routing or distributed task allocation), learning agents do not have access to the global reward signals, because they can not be computed in real-time. Even when they are available for some systems, they are usually not specifically tailored to individual agents' performance and are not good feedback for agents' learning. In this work, we leverage the significance of shared tasks and use the reward function to capture value of tasks from a partially global perspective instead of a local perspective.

In partially observable environments, agents learn *stochastic policies* [Bernstein et al., 2000] that map each state to a set of probability distributions over actions. Stochastic policies can cope with the uncertainty of observations in certain degree and perform better than deterministic policies in partial observable environment. The construction by hand of such adaptation policies is a difficult and time-consuming process. Three of the difficulties in developing such policies are (a) adaptation may affect the activities of other agents, which means that agents need to either implicitly or explicitly coordinate their adaptation policies; (b) agents may need to know information about the state of other neighboring agents in order for their local policies to take appropriate action choices, which means there could be additional communication overhead. (c) The search space for each agent may become substantially large that makes policy learning computationally challenging. We present a decentralized learning algorithm that leverages smart state space expansion and conflict resolution to address the above challenges.

1.1.3 A Real-world Application

For my dissertation work, I will study the above described intellectual questions in the context of NetRads, a real-world application for tracking emerging weather phenomena (e.g., rotations). The NetRads system [Krainin et al., 2007] [Zink et al., 2005] is a network of adaptive radars controlled by a collection of Meteorological Command and Control (MCC) agents that determine for each radar where to scan based on emerging weather conditions. The NetRads radar is designed to quickly

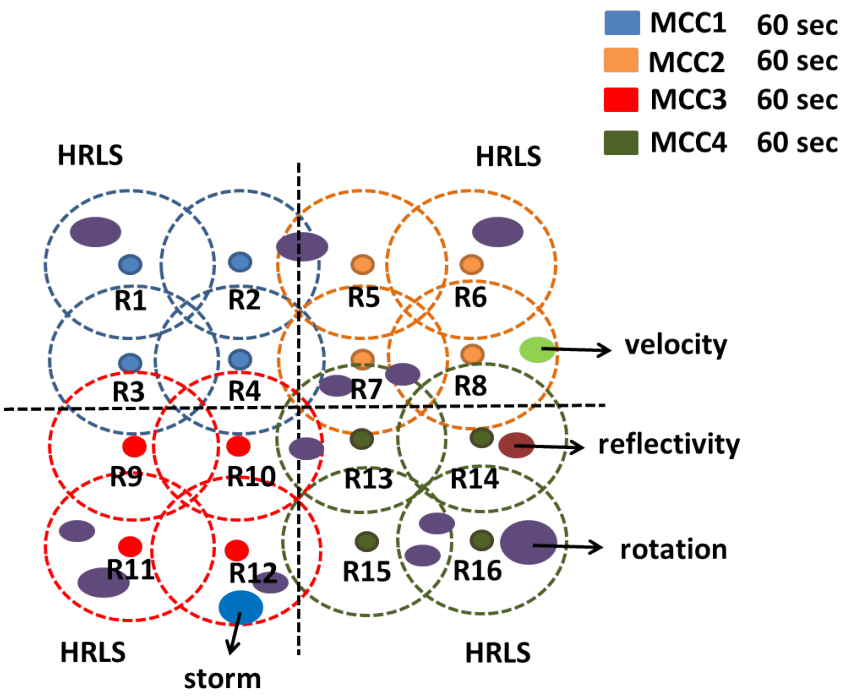


Figure 1.1: The *MCCs* experience homogeneous weather scenarios.

detect low-lying meteorological phenomena. The MCC agent can manage multiple radars simultaneously, where each radar belongs to exactly one MCC. The MCC agents have partial knowledge of their environments (they observe task set, neighbors, data correlation and etc. with limited scope). Figure 1.1 is the example of MCC-Radar configuration, where each MCC controls four radars. Each radar has a scanning area represented by a circle and may have overlapping scanning areas with other radars that demonstrates the environment of partial observability. The key terms used in this work are defined as follows.

Heartbeat: The time allotted to the radar and its control systems for data gathering and analysis of tasks is known as a *heartbeat*. Each NetRads agent has two choices of heartbeat: 60 seconds and 30 seconds. A shorter heartbeat allows the system to respond more rapidly to closely track the quickly evolving weather phenomena. A longer heartbeat allows the system to scan with more resolution. In Figure 1.1, all the MCCs have a 60 second heartbeat.

Task: Each radar scanning *task* in the NetRads system has a position, a velocity, a radius, a priority, a preferred scanning mode, and a type. Tasks are defined as four types of weather phenomena (distinguished by different colors in Figure 1.1): storm, rotation, reflectivity and velocity. Each of these types has its own distributions for the characteristics mentioned above. The differences between these tasks are the size (e.g., storms occupy a much larger area than rotations) and the elevations at which a radar must scan the task to obtain useful information (e..g, storms must be scanned at the lowest four elevations, but rotations must be scanned at the lowest six). Tasks may be either *pinpointing* or *non-pinpointing* (described later). The *utility* of a task from a single radar is the priority of the task multiplied by a factor meant to represent the quality of the data that would result from the scan. The priority of the task is specified by experts in the field such as meteorologists while quality of the scan represents *how well* a particular portion of the atmosphere is sensed by a given radar configuration. For each task t_i, the utility is defined as:

$$u(t_i) = d(t_i) \times q(t_i) \tag{1.1}$$

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 $q(t_i) : t_i \times (s_1, s_2, ..., s_n) \rightarrow r \in \Re$, 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.

Neighbor: Two MCCs are *neighbors* if their radars share overlapping scanning areas (In Figure 1.1, each MCC has three neighbors).

Data Correlation: *Data Correlation* occurs when radars belonging to different MCCs can observe the same weather phenomenon (such weather phenomenon are on the overlapping areas of radars) and thus the interdependency and need for communication among these MCCs would increase. 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. *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, $D_j \in \{High, Medium, Low\}, j = 1, 2, ..., n$. I abstract the *degree of data correlation* into three qualitative categories to decrease the total number of explored states in the search space. When radars belonging to different MCCs share data (especially data about shared pinpointing tasks), they are more interdependent and so the communication among these MCCs would increase. In this work, I assume the value to be $High$ if the percentage of pinpointing tasks between two MCCs is equal or more than 70%; 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 $Medium$. In this work, the values (70% and 30%) are set manually to categorize the three levels of degree of data correlation.

Weather Scenario: *Weather scenario* is the term used to qualitatively abstract the weather environments that NetRads is experiencing at a particular time period. For example, *High Rotation Low Storm* (HRLS) is one type of weather scenarios where the number of rotations is significantly larger than the number of storms in a series of heartbeats. The threshold of "significantly larger than X" is defined in a consistent way as "more than three times compared with X". This qualitative abstraction allows for a substantial reduction in the search space. In Figure 1.1, all the MCCs experience the same type of weather scenario in a simplified environment. In a more complicated, heterogeneous and realistic environment, different parts of the system may encounter different types of weather scenarios. In this work, I will learn individual policies

for the heterogeneous weather scenarios that the NetRads system is experiencing. The performance of my approach is evaluated in three different classes of weather scenarios: *High Rotation Low Storm* (HRLS), *Low Rotation High Storm* (LRHS), and *Medium Rotation Medium Storm* (MRMS). The performance is evaluated based on these three general classes since each class stresses the system in different ways.

HRLS is the weather scenario where the number of rotations is significantly larger than the number of storms in a series of heartbeats (e.g. lots of rotation phenomena move in followed by a few storm phenomena, and then followed by lots of rotation phenomena). 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 distributions is very coarse, it has been found that this level of detail is sufficient to generate policies that can improve the system performance significantly.

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. I define a set NS_i which consists of the *neighborhood scenarios* MCC_i might encounter based on the degrees of data correlation 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 (V_{hb}) and the number of its current radars (V_{radar}) involved in the data correlation with MCC_i. $f_j (j = 1, 2...N)$ is defined as (V_{hb}, V_{radar}), in which $V_{hb} \in \{30seconds, 60seconds\}$ and $V_{radar} \in \{0, 1, many\}$[3]. "$many$" means more than

[3]In the NetRads domain that is evaluated, 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 V_j is categorized into the three buckets to abstract and differentiate V_j without the state space blowing up.

one radar is involved in the data correlation. The qualitative value "*many*" is used to simplify the description of MCC's relation with its neighbors so as to reduce the number of different feature sets. In Figure 1.1, from the view of MCC_2, it is in $NS_2 = \langle(60seconds, 1), (60seconds, 0), (60seconds, 1)\rangle$ which means that MCC_2 has three neighbors (MCC_1, MCC_3 and MCC_4): MCC_1 has the 60 seconds heartbeat and 1 radar involved in the data correlation with MCC_2; MCC_3 has the 60 seconds heartbeat and no radar involved in the data correlation with MCC_2; MCC_4 has the 60 seconds heartbeat and 1 radar involved in the data correlation with MCC_2.

Meta-level control

As discussed earlier,two levels of control are associated with agents' iterative three-step closed loop: deliberative and meta-level control [Cox and Raja, 2008]. The lower control level is *deliberative control*, 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. As Figure 1.2 shows, deliberation-level and domain-level are two separate but related processes; meaning 100% of resources for domain-level are also used for deliberation-level.

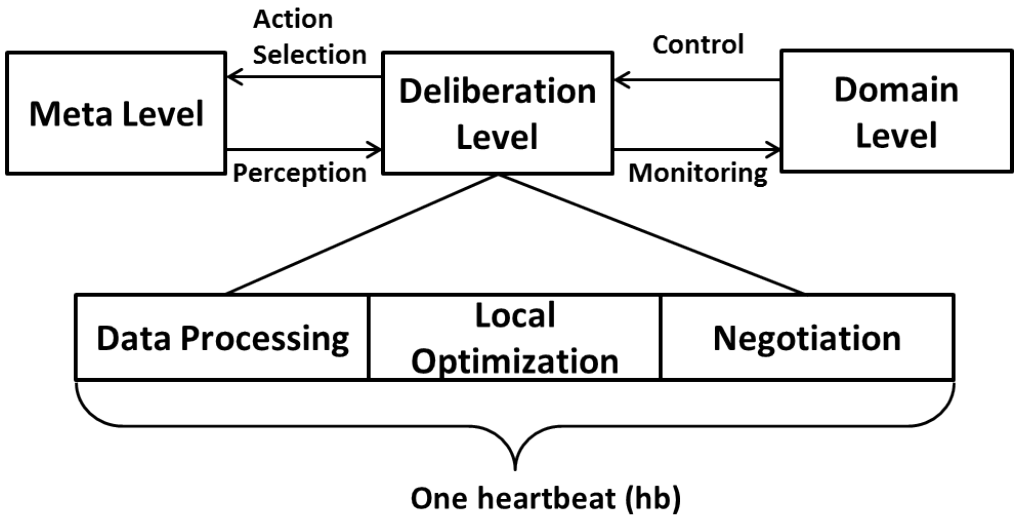


Figure 1.2: Heartbeat with three deliberative-level phases.

In the NetRads application, the domain action at each heartbeat would be the radar

scanning. The deliberative action [Krainin et al., 2007] at each heartbeat would involve each MCC spending some initial time in processing the radar data obtained during the last heartbeat, then performing a local optimization to determine the configuration of the radars under its control, followed by negotiation rounds of alternating communication and recomputation of the local configuration.

These phases are called: *Data Processing*, *Local Optimization* and *Negotiation* respectively (as Figure 1.2 shows). 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 scanning tasks to be assigned to each of its radars to maximize the sum of the task utilities gained for that heartbeat. In *Negotiation*, the MCC communicates with its neighboring MCCs to modify its local optimization to successfully complete joint tasks and to avoid redundant scanning of the same area. The goal of the NetRads is to maximize the sum of the utilities of the tasks scanned in each heartbeat.

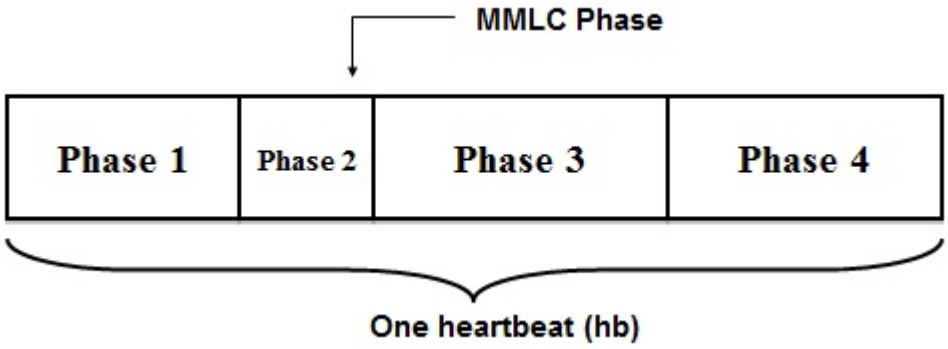


Figure 1.3: Adding MMLC phase in the heartbeat.

Although the types of deliberative actions [Krainin et al., 2007] are well-defined, just sticking to these deliberative actions is not sufficient. Deliberation is computationally expensive and it is important to reason about deliberative choices. A new set of actions are introduced that allow the agent to be smarter about its deliberations

without itself consuming too many resources.

In this dissertation, *multiagent meta-level control* (*MMLC*) is introduced to the NetRads system 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.

Each MCC agent is augmented with a new meta-level phase called *MMLC* that learns policies to handle the coordination of MCC agents and guide the deliberative-level actions in *Local Optimization* and *Negotiation*. Although I apply and evaluate the learning and conflict resolution algorithms to meta-level control in this work, the approach is also applicable to the deliberative level. Each heartbeat is now split up into four phases (as Figure 1.3 shown).

Questions to be addressed

At the highest level, the question I plan to address is the following: "*How does 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 meta-level adaptations:

1. How to re-organize the sub-networks of radars under each agent to minimize the time required for agents to negotiate with their neighboring agents?
2. How to adjust the system heartbeat (how often a radar processing strategy is updated) to adapt to changing weather conditions while balancing response timeliness and scan accuracy?

The intuition behind identifying these meta-level issues is that it is preferable that radars with large data correlation be allocated to the same MCC to reduce both the amount of communication and the time for 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-level control may decide to use a shorter heartbeat to allow the system to respond more rapidly so as to closely track the quickly evolving weather phenomena but with less resolution.

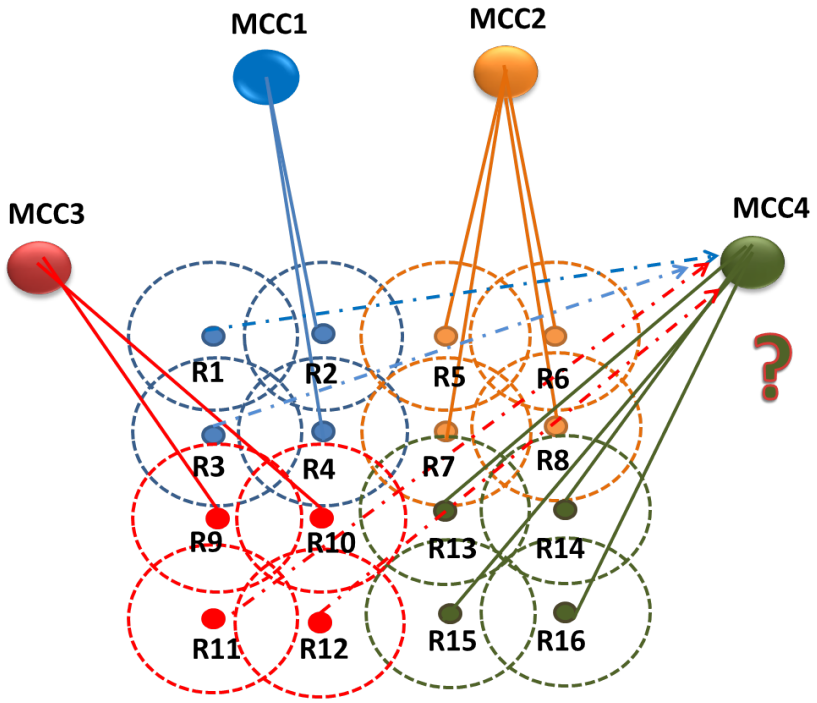


Figure 1.4: An LRC exists when both MCC_1 and MCC_3 move radars to MCC_4.

There are two types of meta-level actions in *MMLC* phase: (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 action of *Radar Reorganization* is more complicated than that of *Heartbeat Adaptation*. It includes domain actions that deliberate about the type of radar movement and the direction of radar movement. 'Heavy Move' and 'Light Move' are two different types of radar movement (will be discussed in Chapter 3.2.2). The former moves a large amount of radars to neighbors while the later moves few to decrease the load of the MCC. For example, one action for *Radar Reorganization* could be 'Heavy Move(MCC_1 to MCC_2)' which means MCC_1 applies a 'Heavy Move' and the radars are moved to MCC_2 (the direction of radar movement). The *Heartbeat Adaptation*

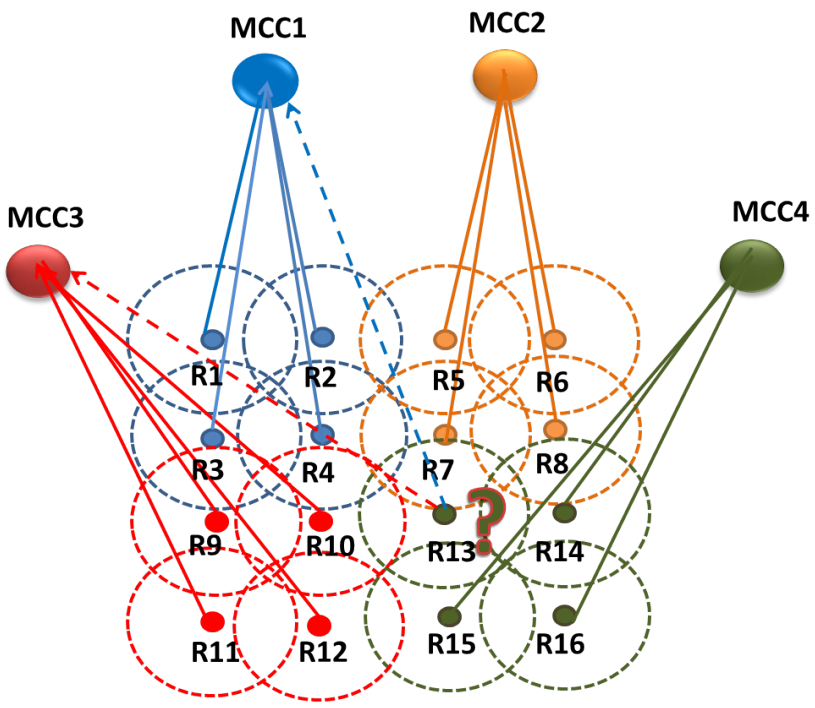


Figure 1.5: An SRC exists when both MCC_1 and MCC_3 require the control of R_7.

action has two domain level action choices: 'Use 30 seconds heartbeat' and 'Use 60 seconds heartbeat'. Simon [Simon, 1976] argued that deliberation time cost should be applied to MMLC as well, since an agent is not performing rationally if it fails to account for the overhead of computing a solution. 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.

In this work, MCCs learn the policies that include these two types of meta-level actions. The *horizon* of the policies for the NetRads is three heartbeat periods. This horizon is defined after manually examining the behavior of the domain in various scenarios. If the horizon is too short, it triggers meta-level control too frequently which increases the cost of decision making and affects performance. On the other hand, a horizon that is too long may result in meta-level control policies that are obsolete for the latter part of the horizon, given the dynamic nature of the environment. In future work, I plan to investigate the effectiveness of learning the horizon. The meta-level

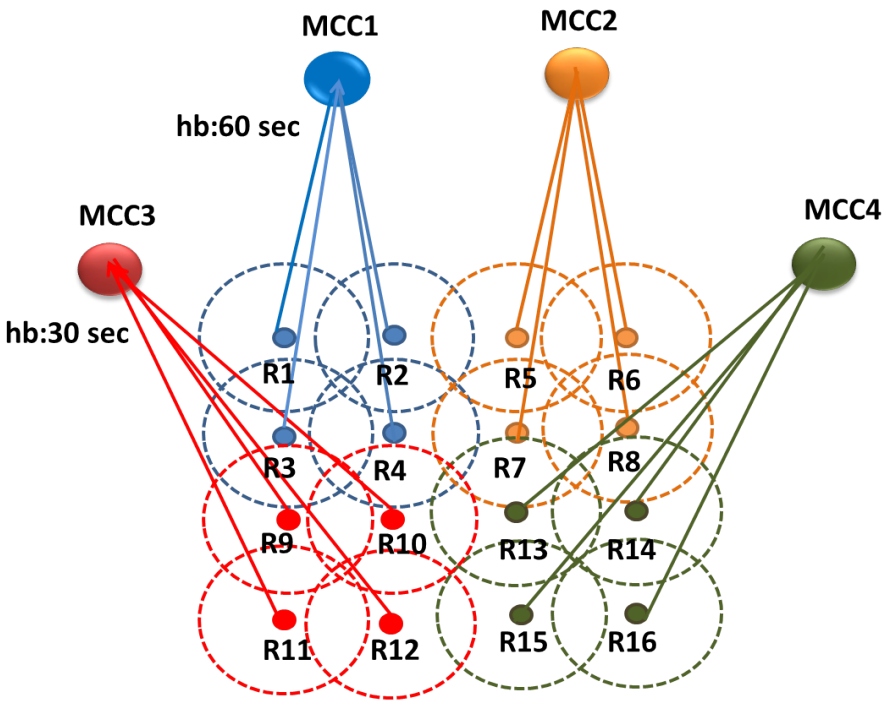


Figure 1.6: An IHC exists when MCC_1 and MCC_3 have different heartbeats.

action is defined as an abstract representation of real action sets. A *detailed action* is an instantiation of an meta-level action. In Figure 1.1, a detailed action of the meta-level action 'Move less than 10% of the radars from MCC_1 to MCC_2' could be 'Move Radar R_1 from MCC_1 to MCC_2'.

The concept of a *conflict* in my work is defined as incompatibilities among two or more agents' local policies for such reasons that agents compete for the same shared resource, agents fail to balance the load of the whole multiagent system and agents are not synchronized in communication. The following types of conflicts among agents' detailed actions associated with meta-level actions are defined as: *Local Radar Conflicts (LRC)* (as Figure 1.4 shows) refer to situations in which an agent receives radars from two or more neighboring agents simultaneously and thus it is overloaded [4]; *Shared Radar Conflicts (SRC)* (as Figure 1.5 shows) are inconsistencies that

[4]The load of each MCC is measured based on the amount of data it needs to process. The number of radar it controls as well as the number of tasks (especially pinpointing tasks) in the overlapping areas contribute to the load.

may arise when two or more agents attempt to move the same radar(s); and *Inconsistent Heartbeat Conflicts (IHC)* (as Figure 1.6 shows) occur when two neighboring agents have different heartbeats and have to communicate with each other during the *Negotiation* phase. The three types of conflicts have different degrees of importance to solve based on the influence they have on the system performance.

1.1.4 Motivating Example

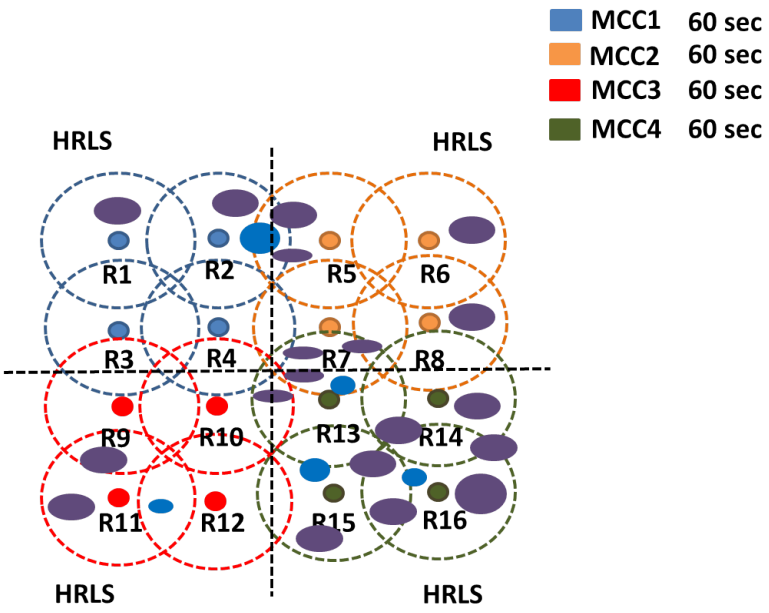


Figure 1.7: The initial network topology. Radar R_2 has a large data correlation with R_5; R_{13} has a large data correlation with R_7. Many scanning tasks occur in the region of MCC_4.

I use a motivating example to show the implementations of these two types of meta-level actions. In Figure 1.7, a large number of shared tasks occur in the overlapping areas between R_2 and R_5; R_{13} and R_7. For the sake of local optimization, MCC_2 could decide to move radars R_5 and R_6 to its neighbor MCC_4; meanwhile from the perspective of MCC_3, it could also decide to move radar R_{10} to its neighbor MCC_4 to achieve local optimization. However, this results in a L-RC which makes MCC_4 overloaded. An optimal policy for each MCC is needed to minimize conflicts and at the same time to reach globally optimal solution (the

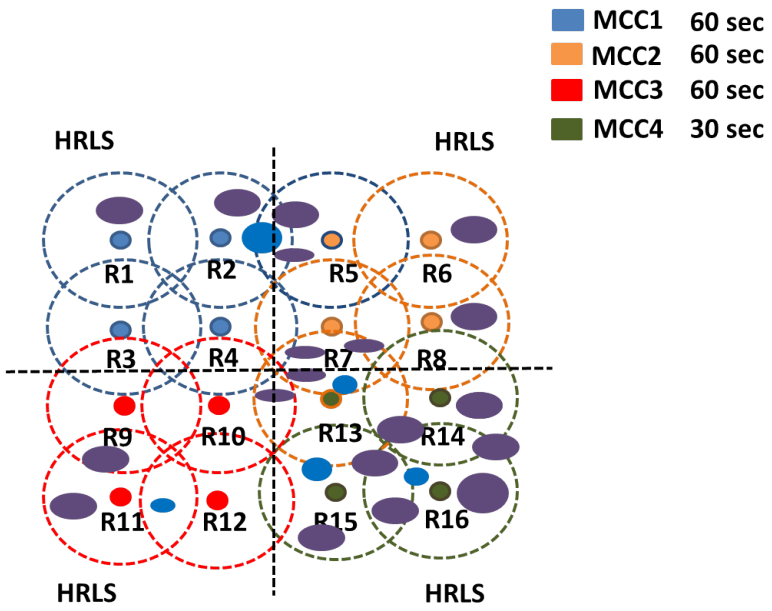


Figure 1.8: The resulted network topology after meta-level actions (radar reorganization and heartbeat adaptation) are executed.

overall performance is maximized). In this example, one optimal policy for each MCC would be: $\{MCC_1 : \emptyset\}$; $\{MCC_2 : \text{“}Move\ R_5\ to\ MCC_1\text{”}\}$; $\{MCC_3 : \emptyset\}$; $\{MCC_4 : \text{“}Move\ R_{13}\ to\ MCC_2\text{”}, \text{“}Use\ 30\ second\ heartbeat\text{”}\}$. By making such changes in MCC-radar associations, the system reduces the cost of communication and the time spent on negotiation among MCCs. Also it is preferable for MCC_4 to use a shorter heartbeat (30 seconds) to respond more rapidly and to closely track the quickly evolving tasks in its region. Figure 1.8 is the network topology resulting from such a reorganization.

1.2 Assumptions

The agents are cooperative and will prefer alternatives which increase social utility/quality even if it is at the cost of decreasing local utility. The overall objective of the system or agent is to maximize the utility generated over some finite time horizon. MMLC is time constrained and can use only up to 10% of the total deliberation time. The neighbors will always respond, and their responses will be always received. The agents will all know when to terminate negotiations and do so at the same time by

synchronizing their clocks.

The *learning stage* of this work describes the agent's off-line learning process where it adapts its behavior to improve performance. During the learning stage, the complexity of the real state is handled by using the *meta-level state* that captures the important qualitative state information relevant to the meta-level control decision making process. The learning process is also sped up by defining the meta-level action as an abstract representation of real actions and each meta-level action has multiple possible detailed actions associated with it. A *detailed action* is an instantiation of an meta-level action. For instance, the meta-level action "Move less than 20% of the radars from MCC_1 to MCC_2"[5] could have the following detailed actions: "Move 1 radar from MCC_1 to MCC_2" or "Move 2 radars from MCC_1 to MCC_2". When an agent takes an action, the "action" that is really implemented is a detailed action. *Execution stage* in this work is denoted as the agent's real-time execution process when it chooses and implements the appropriate policy.

In this work, conflicts among agents' actions are resolved at both learning and execution stages so as to compute approximate optimal policies. Resolving conflicts at only execution stage is not sufficient. Learned policies implicitly coordinate agents' behaviors, and so resolving conflicts only at execution stage may break their coordination and result in poor performance.

The key idea in this work is that agents are smart about obtaining contexts and requiring just relevant information to reduce conflicts. Suppose there are conflicts among action choices, agents need to gather more contextual information. In other words, when agents get into problems they acquire more information to reduce conflicts. The more context an agent has about its neighbors, the more coordinated its decisions will be. It is expensive to learn in large contexts so we build local policies making simplistic assumptions about the neighbors. An agent decides locally to take

[5]Suppose MCC_1 has 11 radars associated with.

an action but before executing recognizes if there is a conflict at that point and can then take a different action. In other domains, conflicts could lead to only reduced utility. In NetRads, conflicts reduce quality and conflict resolution has overhead. Our idea is to reduce amount of time/effort involving conflict resolution strategy.

1.3 Summary of Approach

This dissertation seeks to answer the following key question: *How can a network of agents, operating in such complex cooperative domains, effectively learn policies that support agent interactions with other agents and reorganize the underlying network when needed?*

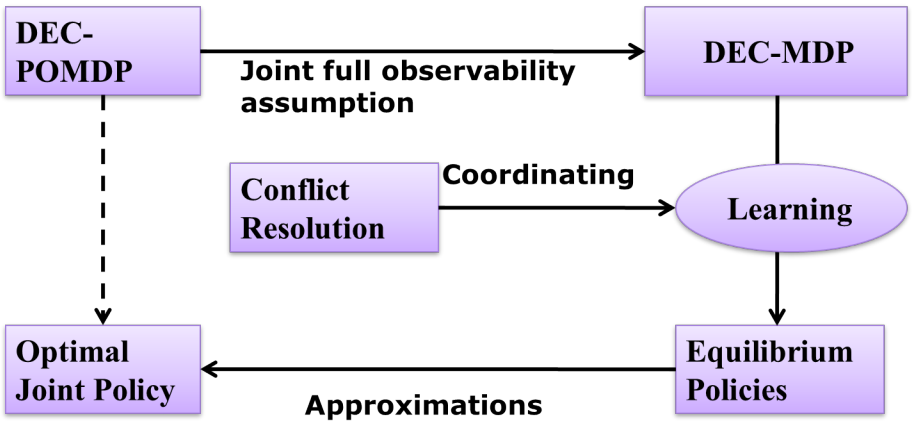


Figure 1.9: An encompassing view of the MARL paradigm.

I model the problem as a decentralized coordination problem and describe a multiagent learning approach that uses decentralized Markov decision processes (DEC-MDPs) [Bernstein et al., 2000] to learn the locally optimal policy for each agent. To address the NEXP-Complete complexity of DEC-MDP, the solution is approximated by using a factored reward function to define the Nash Equilibrium [Osborne and Rubinstein, 1994] [Singh et al., 2000] (as shown in Figure 1.9). I extend a practical MARL algorithm, called *Policy Gradient Ascent with approximate policy prediction (PGA-APP)* [Zhang and Lesser, 2010], to learn stochastic policies for the DEC-MDPs

belonging to individual agents.

The real-world weather scenarios are categorized into different classes by considering the effects they have in the system and the learning process is sped up by learning the policies separately for each class. The exploration costs of DEC-MDPs are decreased by constructing abstract classes of states and actions where instances within a class have similar features.

My dissertation establishes the following hypothesis: Harnessing decentralized learning and conflict resolution helps a cooperative MAS to converge to individual agent policies that effectively improve the global performance.

As part of my adaptive approach to solve this problem, a heuristic rule-based approach to locally change the detailed actions to resolve the conflicts in simplified environments is first presented. The conflicts are categorized and pre-defined rules are used to resolve different types of conflicts between two agents. The empirical results show that this approach gives a performance advantage over an approach that resolves conflicts in a much more ad-hoc manner.

However, the uncertainty and complexity of the environment often lead to very large state space for each agent making decentralized policy learning computationally challenging. I present a decentralized learning approach that is based on the idea of selectively expanding the search space and learning the best policy for each agent by harnessing information about the real-time performance of conflict resolution. The conflicts are assigned different priorities based on the importance for resolution; these priorities are taken into account in the approach. When conflicts resulting from multiple neighboring agents applying their local policies are observed, agents switch to "special" states that augment local policy states with additional non-local state information and learn other actions to take in this specific situation. A mediator-based mechanism is used to resolve conflicts in a partially global perspective. It breaks the conflicts into smaller sets, partially centralizes and solves each set from

a local view in a limited time period. Experimental results show that this approach achieves good performance on system utility and conflict resolution by unrolling a small fraction of the whole search space.

The above two approaches are designed to resolve conflicts from a local or partially global perspective and survive in the scenarios where the dependency among agents is not global and when we can believe that the interaction is limited to subset of agents. To handle scenarios that the dependency among agents is highly constrained and the interaction is globally distributed, we need distributed constraint optimization algorithms to resolve conflicts and compute solutions from a global perspective. Distributed constraint optimization problems (DCOPs) are a broad family of problems that a group of agents control the state of the variables in the system, having the purpose to maximize the global reward for satisfying all the constraints. I model the conflicts resolution problem as a DCOP and use a state-of-the-art algorithm that solves DCOP, called Max-sum [Farinelli et al., 2008], to coordinate agents' actions. Empirical results show that the approach produces optimal solutions that minimize the severity of conflicts at the global level.

1.4 Overview of Contributions

The contributions of this dissertation are:

1. A novel MAS framework for decentralized learning so that it can efficiently support agent interactions and reorganize the underlying network when needed.
2. Efficient techniques to decrease the exploration costs of DEC-MDPs by constructing abstract classes of scenarios/ states/actions where instances within a class have similar features.
3. Improving learning rates by categorizing different weather scenarios and learning policies for each MDP of each weather scenario.
4. Leveraging the significance of shared tasks in the NetRads domain and using the reward function for the reinforcement learning to capture value of tasks

from a partially global perspective instead of a local perspective.

5. Coordination of DEC-MDPs that use multiagent reinforcement learning to learn joint polices in a decentralized fashion and incorporate an adaptive conflict resolution approach. The adaptive approach involves two algorithms that resolve conflicts from a local and partially global perspective as well as a DCOP algorithm that resolves conflicts from a global viewpoint.
6. A rigorous study of the interplay between two popular theories used to solve multiagent problems, namely decentralized Markov decision processes and distributed constraint optimization.
7. A rigorous empirical study that spans both homogeneous and heterogeneous (more complex) environments to show that the adaptive approach helps to resolve conflicts and improve the overall performance.

1.5 Outline

The structure of this dissertation is as follows. In Chapter 2, I give an overview of related work in multiagent systems, multiagent learning frameworks and multiagent reinforcement learning algorithms. I also discuss research in DCOPs, meta-level control and NetRads application domain as well as compare other people's work to my work.

Chapters 3, 4 and 5 comprise the body of the dissertation. Chapter 3 describes the DEC-MDP framework developed for my learning approach. To describe the framework, the idea of abstraction is first presented, especially emphasizing on the abstraction of states and actions. This is followed by a formal description of other critical characteristics in the framework that include transition function, factored reward function, Nash Equilibrium and stochastic policy.

In Chapter 4, research in multiagent reinforcement learning is described. I present a MARL algorithm for local policy learning as well as the control flow that coordinates DEC-MDPs and use this learning algorithm to learn joint policies in a decentralized

fashion. The performance of this learning algorithm provides a sanity check on the effectiveness of the DEC-MDP framework without conflict resolution.

Chapter 5 begins with a discussion about conflicts that may occur between agents' local policies. I first present a categorization of the conflict types and priorities, followed by a description of three incrementally more complex algorithms to resolve conflicts and thus learn more effective policies for the whole system. The first conflict resolution algorithm uses heuristic rules to locally resolve simple conflicts; the second algorithm selectively expands agents' state space and learns the decentralized policy by harnessing information about the real-time performance of conflict resolution. It uses a mediator-based mechanism to resolve conflicts from a partially global perspective. The final conflict resolution algorithm uses DCOP algorithm to resolve conflicts from a global perspective. Experiments comparing and describing the viabilities of these algorithms in different system environments are presented.

The dissertation concludes with Chapter 6, in which the main results and applications of this research and directions for future research are discussed.

CHAPTER 2: RELATED WORK

In this dissertation, I mainly deal with decision-making problems involving multiple agents, and address the question how the agents operating in complex cooperative domains, can effectively learn policies that support agent interactions and conflict resolution. The approach is applied at the meta-level in a real-world radar tracking system. In this chapter, I will discuss the current state of the art as it relates to the research in this dissertation. This includes research in multiagent systems, multiagent learning frameworks, multiagent reinforcement learning, MDP unrolling, DCOPs and distributed algorithms, meta-level control and NetRads application domain and compare it to my work.

2.1 Multiagent Systems

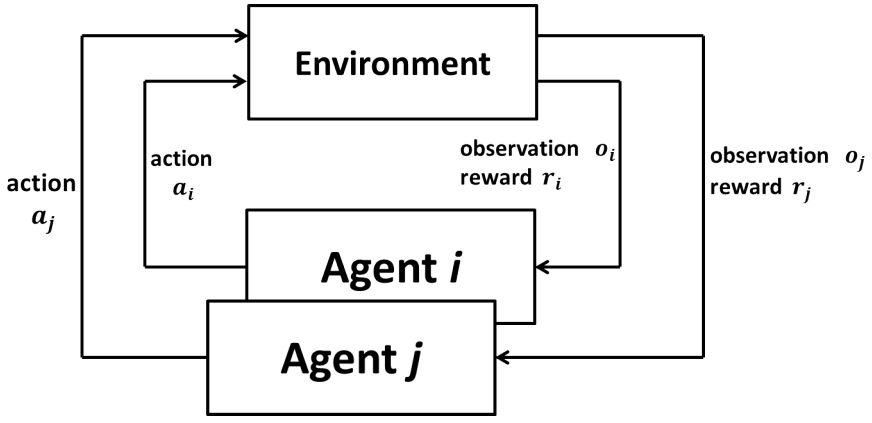


Figure 2.1: A representation of two agents interacting with their environment.

A recent research effort on distributed artificial intelligence is to apply agent technology to intelligent network management and data harvesting. Agents are autonomous entities that receive sensory inputs from the environment and then act on it using their effectors based on the knowledge they have of the environment [Russell and Norvig, 2003]. A multi-agent system (MAS) consists of multiple agents

which are all executing actions and influence their surrounding environments [Sycara, 1998] [Weiss, 1999]. A MAS allows for the distribution of knowledge, data, and resources among individual agents and its modularity supports the development and maintenance of complex highly reliable systems [Wooldridge, 2002]. Figure 2.1 shows a representation of different agents interacting with their environment. In principle, the agents in a MAS can have different, even conflicting goals. However, in this dissertation I am interested in cooperative MASs in which the different agents form a team with the same goal [Pynadath and Tambe, 2002]. This is an important topic in distributed AI, since many large-scale applications, for example, sensor networks and task allocation, are formulated in terms of functionally or spatially distributed entities. Collaboration enables the different entities to work more efficiently and to complete activities they are not able to accomplish individually. The use of a MAS has the following advantages [Sycara, 1998]:

- A MAS can speed up the operation of a system because the agents can perform the computations in parallel.
- A MAS usually has a high degree of robustness. In a single-agent system, the entire system is crashed when a single failure happens. In a MAS, if one or several agents fail, the system will still be operational because the remaining agents can take over the workload.
- A MAS is easier to scale up. Adding new functionality to a monolithic system is often much more difficult.
- A MAS has less communication bandwidth requirements since processing is located nearer the source of information and facilitating real-time responsiveness as processing, sensing, and effecting can be co-located.

A major challenge is to formalize these types of problems and construct solutions to coordinate the different behaviors of the agents. Application domains include sensor networks [Lesser, 2003] [Modi et al., 2003], network routing [Dutta et al., 2005],

robotics [Low et al., 2011], energy [Chalkiadakis et al., 2011] and security [Shafi and Merrick, 2011]. Next, I describe several other fundamental characteristics of a MAS in more detail.

Dynamic environment

In most single-agent systems, the environment is assumed to be static, which means that the transition and reward function do not depend on the time step t. However, in a MAS, when other agents are part of the environment the new state and received reward for one agent also depend on the actions selected by the other agents. The environment becomes dynamic from the view of a single agent as a result. In NetRads, when one MCC chooses to do a "Heavy move" for radar reorganization, the outcome depends on the behavior of other neighboring MCCs. The other neighboring MCCs might anticipate the "Heavy move" of this MCC, but it is also possible that they ignore the action of this MCC. The behavior of other agents can change over time, resulting in a dynamic environment. Dynamic environments are more difficult to handle than static environments since the same action can have different effects based on factors an agent is not able to influence. This might lead to oscillated behavior [Boutilier, 1999], and therefore requires solution techniques in which the agents actively synchronize and coordinate their behaviors.

Homogeneous and heterogeneous agents

In a MAS, agents are either *homogeneous* or *heterogeneous*. Homogeneous agents are constructed in the same way and have identical capabilities [Weiss, 1999]. For example, hardware robots manufactured by the same factory process are homogeneous agents. On the other hand, heterogeneous agents have different designs and different capabilities. For example, two soccer robots are heterogeneous when they have different roles on the field and traverse the field with different velocities. In this research, I focus on homogeneous agents.

Communication

In a MAS, *communication* can help a team of agents to improve their performance. In the extreme case, the agents are able to communicate instantaneously to all agents for free without limitations in the number of messages. Then, it is in principle possible to solve the system as one big single agent: one agent collects all the information based on observations, solves the complete problem using single-agent learning techniques, and informs each agent which action it should take. In this case, each agent obtains perfect knowledge about the current situation, and therefore is able to model the complete problem by itself, and select the action corresponding to its own identity.

In real applications, however, communication is restricted. For example, communication might not be available because of failing connections or spatial constraints. Furthermore, communication is often delayed. In order to model the drawbacks of communication, the sending of a message is sometimes associated with a cost, for example, in the form of a negative reward [Pynadath and Tambe, 2002]. In NetRads, it is critical to control the communication between agents to a reasonable degree at the meta level. Too much communication may result in increased time spent at the meta level, thus leading to less time at the deliberative phases (Phase 3 and Phase 4 in NetRads). On the other hand, sub-optimal meta-level policy is made mainly due to insufficient communication between neighboring agents.

2.2 Meta-level Control

In this work, I focus on the MAS framework and investigate in the meta-level phase in a real-time application. In complex environments, autonomous systems generally require the ability to reason about resource allocation to computation at any point in time. Doyle's 'rational psychology' project [Doyle, 1983] is based on the idea that computations, or state changes, are also actions to be reasoned about. He used the idea of bounded rationality in the context of beliefs, intentions and learning. Horvitz [Horvitz, 1988] also studied rational choice of computation in the context of designing intelligent systems.

The basic idea of bounded rationality arises in the work of Simon with his definition of procedural rationality [Simon, 1982]. Simon's work has addressed the implications of bounded rationality in the areas of psychology, economics and artificial intelligence. He argues that people find satisfactory solutions to problems rather than optimal solutions because people do not have unlimited processing power. 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. [Russell et al., 1993] 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. Schut and Wooldridge [Schut and Wooldridge, 2001] observed that a Markov Decision Process (MDP)-based model toward decision making is most similar to the bounded optimality model. Cox [Cox, 2005] provided a review of metacognition research in the fields of artificial intelligence and cognitive science.

Meta-level control in complex agent-based settings was first explored in [Raja and Lesser (2007)], where Raja and Lesser developed a sophisticated architecture that could reason about alternative methods for computation, including computations that handled simple negotiation between two agents. 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. Cox and Raja [Cox and Raja, 2008] [Cox and Raja, 2011] presented in plain language and simple diagrams a description of a model of metareasoning that mirrors the action-selection and perception cycle in first-order reasoning. In my work, meta-level control is used in a more complex application of real-world system to support agent interaction and network reorganization. I abstract the meta-level states and actions to bound the size of policy set and make

the problem tractable.

Carlin and Zilberstein [Carlin and Zilberstein, 2011] considered a decentralized setting, where multiple agents are solving components of a larger problem by running multiple anytime problem solving algorithms concurrently. They proposed a formal model, where meta-reasoning is used to monitor the progress of the anytime algorithms and decide when to stop deliberation. In the decentralized monitoring problem (DMP), Carlin and Zilberstein assume that the meta-level actions are known to each agent (having 4 options); while in the NetRads domain, the set of meta-level actions is more complicated and exponential in size and is learned through DEC-MDPs. Additionally, the cost defined for each meta-level action in the DMP application is constant. However, in the NetRads domain, the cost of meta-level control is dynamic and the computation is sophisticated of radar re-organization. In the DMP application, meta-level policies are used to monitor locally/globally other agents or continue/stop the computation of agents to achieve optimal solution for global quality. In this work, meta-level policies are learned and used to guide the actions at deliberative level.

Kennedy [Kennedy, 2010] 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 my knowledge, there is very little work done in the area of exploring the coordination of meta-level control parameters across agents and I plan to exploit that idea in this work.

2.3 Multiagent Learning Frameworks

At the meta-level phase, it is important for each agent to learn optimal policies and make smart decisions that help achieve the overall performance. In a sequential decision-making problem an agent repeatedly interacts with its environment and tries to optimize a performance measure based on the rewards it receives. It is difficult

to determine the best action in each situation because a specific decision can have a long-term effect and its particular outcome often depends on the future actions that will be performed. Referring back to my NetRads example, it is impossible for an MCC agent to reverse its decision for radar reorganization within a heartbeat.

I assume the sequential decision-making problems considered in this dissertation all obey the Markov property. This implies that the current situation provides a complete description of the history, and previous information is irrelevant for making a decision. Furthermore, I assume that the decision-making agent is both autonomous and rational. Autonomous means that the agent is capable of making decisions on its own, and thus without the guidance of a user. Rational means that the agent should select actions to maximize a given performance measure based on the available information received from its sensors and its knowledge about the problem.

In this dissertation I mainly deal with sequential decision-making problems involving multiple agents, and address the question how the agents in a group can take the right decision in a decentralized fashion for any given scenario. I restrict my attention to cooperative multiagent systems in which the agents have to work together in order to achieve a common goal, that is, optimize the given performance measure. This differs from self-interested approaches [Shapley, 1953] [Littman, 1994] in which each agent tries to maximize its own performance (e.g., in two-player games).

In this section, I describe different models for sequential decision making. I review the single-agent Markov decision model and then extend it to incorporate multiple agents. Stochastic games and other multiagent models are discussed.

2.3.1 Markov Decision Processes (MDPs)

Model: Markov Decision Processes (MDPs) [Puterman, 1994] [Bertsekas and Tsitsiklis, 1996] are the foundations for much of the research in the single agent learning. They provide a formal framework to model single agent decision-making problems with uncertainty. In this sub-section, I will review the MDP model and its corre-

sponding solution concepts.

Definition 1. A *Markov decision process* is defined by a tuple $\langle S, A, T, R \rangle$, where

- S is a set of states,
- A is a set of actions,
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition function,
- $R : S \times A \rightarrow \Re$ is a reward function.

The transition function defines a probability distribution over next states as a function of the current state and the agent's action. The reward function defines the reward the agent receives after taking an action in a state. Both the transition functions and reward functions satisfy the Markov property, that is, the next state and the reward solely depend on the current state and action, and not on the history of states and actions [Puterman, 1994]. An agent interacts with its environment through the alternation between perception and action. The agent observes the state s^t at time t, and selects an action a^t. The agent then receives the reward r^t, and observes the next state, s^{t+1} with the probability $T(s^{t+1}|s^t, a^t)$. A sequence

$$s^0, a^0, r^0, s^1, a^1, r^1, ...s^t, a^t, r^t, ...$$

refers to an execution trace of an agent.

Solution Concept: For MDP problems, we need to find an optimal *policy* for an agent. A policy $\pi : S \rightarrow A$ is a mapping function that specifies an action $\pi(s) \in A$ in each state $s \in S$. Note that the policy for MDPs defined here is deterministic, which means the agent always chooses a particular action for a state. We will define stochastic policies later, which are more relevant for stochastic games. An *optimal policy* for an MDP is defined as the policy that maximizes the function of the rewards received by executing the policy over a potentially infinite horizon. There are two types of functions of the rewards: cumulative reward and discounted reward.

a) Cumulative Reward: For open systems that run for a very long time, we are usually interested in maximizing the cumulative reward over the time. The goal of

the agent is to select actions that optimize a performance measure related to the received rewards. The expected *cumulative reward* R^t collected from time step t is defined as:

$$R^t = r^t + r^{t+1} + r^{t+2} + r^{t+3} + ... + r^T = \Sigma_{k=t}^{T} r^k, \tag{2.1}$$

in which T is the final time step. This measure applies to episodic tasks in which the interaction with the environment is divided into episodes. After T steps, or when a terminal state is reached, the episode ends and the system resets to a starting state.

b) Discounted Reward: In the discounted reward formulation, immediate reward is preferred over future reward. The expected *discounted reward* is specified as:

$$R^t = r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + ... = \Sigma_{k=0}^{\infty} \gamma^k r^{t+k}, \tag{2.2}$$

where $\gamma, 0 \leq \gamma < 1$, is the discount rate. For $\gamma < 1$, rewards received in the near future are favored as more valuable than later received rewards. As γ decreases, this effect is more apparent; in the extreme case $\gamma = 0$ the agent only tries to maximize the immediate received reward. Using this reward formulation, the goal for an agent is to find an optimal policy π^* that maximizes the discounted future reward for all states. If the state transition function T and the reward function R is known, the optimal policy can be calculated using a standard family of algorithms, e.g., value iteration [Bellman, 1957] and policy iteration [Howard, 1960]. In this dissertation, I am concerned about how to learn the optimal policy if the transition function and the reward function are not known.

2.3.2 Stochastic Games

Model: In this section, I will focus on stochastic games, which are more interesting for studying multiagent learning. Stochastic games [Shapley, 1953] are a superset of MDPs, which can have multiple agents and multiple states. A stochastic game is a dynamic game with probabilistic transitions played by one or more players, within a sequence of stages. At the beginning of each stage, the game is in some state. The

players choose actions and each player receives a payoff that depends on the current state and the chosen actions. The game then transits to a new random state. The procedure is repeated and the players continue to play for a finite or infinite number of stages.

Definition 2. An n-agent *stochastic game* is defined by a tuple $\langle A_1, ..., A_n, S, T, R_1, ..., R_n \rangle$, where

- A_i is the set of actions available to player i (and $A = A_1 \times A_2 ... \times A_n$ is the joint action space).
- S is a set of states.
- $T : S \times A \times S \to \Re$ is the transition function. $T(s'|s, a)$ is the probability of transiting to the next state $s' \in S$ after a joint action $a \in A$ is taken by agents in state $s \in S$.
- $R_i : S \times A \to \Re$ is the payoff or reward function of player i. Agent i receives an individual reward $R_i(s, a)$ for the joint action $a \in A$ in state $s \in S$.

In stochastic games, there are multiple agents with each agent having an explicit action set. For the agent set, the joint action and the current state determine the individual next state and rewards. It should be noted that each agent has its own independent reward function.

Solution Concept: Like in MDPs, the goal for each player in a stochastic game is to find a policy that maximizes its long-term reward. The reward formulations of cumulative reward and discounted reward also can be applied to stochastic games to quantify the value of a joint policy to each player. For stochastic games, stochastic policies are learned since they can cope with the uncertainty of observations to a certain degree and perform better than deterministic policies in partially observable environments. A *stochastic policy* for play i, π_i, is a mapping that defines the probability of selecting an action from a particular state. Formally, $\pi \in S \times A_i \to [0, 1]$, where $\Sigma_{a \in A_i} \pi(s, a) = 1, \forall s \in S$. I use $\pi = \langle \pi_1, ..., \pi_n \rangle$ to refer to a joint policy for all

the players, with π_i being player i's policy within that joint policy.

A fundamental solution concept in stochastic games is a Nash Equilibrium [Nash, 1950] [Osborne and Rubinstein, 1994]. It defines a joint action $a^* \in A$ with the property that for every agent i holds $R_i(a_i^*, a_{-i}^*)$ for all actions $a_i \in A_i$, where a_{-i} is the joint action for all agents excluding agent i. Such an equilibrium joint action is a steady state from which no agent can profitably deviate given the actions of the other agents. In NetRads, the agents learn policies that converge to Nash Equilibrium.

POIPSG

Peshkin et al. [Peshkin et al., 2000] describe *partially observable identical payoff stochastic game* (POIPSG), the interaction of a set of agents with a Markov environment in which all agents receive the same payoffs and they do not have the identity observation function. In this setting, after each agent performs its action given its observation according to some individual strategy, they all receive the same payoff. The objective is to find a learning algorithm that makes each agent independently find a strategy that enables the group of agents to receive the optimal payoff. The POIPSG model is not applicable to NetRads, since the reward each agent receives is from a partially global perspective and is different from others.

MMDP

When all players have the same reward function, such stochastic games are called multiagent Markov decision processes (MMDP) [Boutilier, 1999]. MMDP is used to model decision-making problems in fully cooperative multiagent systems where all agents individually observe the state of the environment. Each agent has the same utility or reward function. The MMDP model does not apply to NetRads, because: 1) In NetRads, each MCC has its partial observability of the global state. 2) Communication of observations between MCCs during the *MMLC* phase can be very expensive since the *MMLC* phase is time critical (less than 10% of deliberation time).

2.3.3 Decentralized MDP Models

In this dissertation, I am also interested in learning in the concept of cooperative multiagent systems, where a group of agents work together to optimize the global performance. In this section, I will first review the framework of decentralized partially observable Markov decision processes (DEC-POMDP) to model the sequential decision-making problem in cooperative multiagent systems. I then discuss other decentralized MDP models that are derived from the DEC-POMDP model.

DEC-POMDP

Model: In [Bernstein et al., 2002], Bernstein et al. studied decentralized control problems and centralized control problems from the point of 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. The DEC-POMDP model [Bernstein et al., 2002] with n agents is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R}, h \rangle$, where

- $\mathcal{S}$ is a set of states, with a distinguished initial state s^0.
- $\mathcal{A} = A_1 \times A_2 \times \ldots \times A_n$ is a set of joint actions, where A_i is the action set for agent i.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. $\mathcal{T}(s' \mid s, a)$ is the probability of transiting to the next state s' after a joint action $a \in A$ is taken by agents in state s.
- $\mathcal{Z} = Z_1 \times Z_2 \times \ldots \times Z_n$ is a set of joint observations, where Z_i is the observation set of agent i.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function, where $O(z \mid s, a)$ denotes the probability of perceiving joint observation z after executing joint action a and arriving in state s.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \rightarrow \mathcal{R}$ is the reward function. $\mathcal{R}(s, a)$ is the reward for taking action $a \in A$ in state $s \in S$.
- If the DEC-POMDP has a finite horizon, that horizon is represented by a positive integer h.

Sub-classes of DEC-POMDPs can be characterized based on how the global states, transition function, observation function, and reward function relate to the partial view of each of the controlling agents [Goldman and Zilberstein, 2004]. In the simplest case, the global states can be factored, the probability of transitions and observations are independent, the combined observations determine the global state, and the reward function can be easily defined as the sum of local reward functions. In this extreme case I can say that the DEC-POMDP is equivalent to the combination of n independent MDPs.

Solution Concept: Since an agent has no direct access to the current state in DEC-POMDPs, selecting actions based on the current state (as in a MDP) is no longer valid. An agent needs act based on perceived observations. As discussed earlier, stochastic policies can cope with the uncertainty of observations in certain degree and perform better than deterministic policies in partial observable environment. The policy π is defined as: $Z_i \times A_i \rightarrow [0, 1]$ for agent i as a mapping of an observation $z_i \in Z$ to a probability distribution over action A_i. I use $\pi = \langle \pi_1, ..., \pi_n \rangle$ to refer to a joint policy for all the agents, with π_i being agent i's policy within that joint policy.

DEC-MDP

Model: A decentralized Markov decision process (DEC-MDP) is a DEC-POMDP with the restriction that at each time step the agents' observations together uniquely determine the state. The DEC-MDP model [Becker et al., 2004] is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, where

- $\mathcal{S}$ is a finite set of world states, with a distinguished initial state s^0.
- $\mathcal{A} = A_1 \times A_2 \times \ldots \times A_n$ is a finite set of joint actions. A_i indicates the set of

actions that can be taken by agent i.

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the transition function. $\mathcal{P}(s' \mid s, (a_1...a_n))$ is the probability of the outcome state s' when the joint action $(a_1...a_n)$ is taken in state s.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the reward function. $\mathcal{R}(s, (a_1...a_n), s')$ is the reward obtained from taking joint action $(a_1...a_n)$ in state s and transitioning to state s'.
- $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ is a finite set of joint observations. Ω_i is the set of observations for agent i.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow \mathcal{R}$ is the observation function. $\mathcal{O}(s, (a_1...a_n), s', (o_1...o_n))$ is the probability of agents 1 through n seeing observations o_1 through o_n (agent i sees o_i) after the sequence $s, (a_1...a_n), s'$ occurs.
- Joint full observability: the $n-tuple$ of observations made by the agents together fully determine the current state. If $\mathcal{O}(s, (a_1...a_n), s', (o_1...o_n)) > 0$ then $\mathcal{P}(s' \mid (o_1...o_n)) = 1$.

DEC-MDP applies in the following MAS scenarios. In the decentralized view, an agent cannot see other agents' local states and local actions, and has to decide the next local action on its own. Each agent has only a partial view of the system's global state, and different agents have different partial views. Of course, this does not necessarily mean that the agents are isolated. Rather, an important ability of decentralized cooperative agents is their ability to communicate. In [Xuan and Lesser, 2002], the authors view communication as a way of expanding an agent's partial view by exchanging local information not observed by other agents. DEC-MDPs is a problem with the complexity of NEXP-complete.

An n-agent DEC-MDP is said to be *transition independent* [Goldman and Zilberstein, 2004]: that is, the new local state of each agent depends only on its previous local state, the action taken by that agent, and the current external features.

An n-agent DEC-MDP is said to be *observation independent* [Becker et al., 2003]: that is, the observation an agent sees depends only on that agent's current and next local state and current action.

An n-agent DEC-MDP is said to be *locally fully observable*: that is, each agent fully observes its own local state at each step.

An n-agent DEC-MDP is said to be *reward independent*: that is, the overall reward is composed of a function of the local reward functions, each of which depends only on the local state and local action of one of the agents. This function is such that maximizing each of the local reward functions individually maximizes the function itself.

ND-POMDP

Due to the intractability of optimally solving general DEC-POMDPs, research has focused on restricted versions of DEC-POMDP that are easier to solve yet rich enough to represent many practical applications. Networked Distributed POMDP (ND-POMDP) [Nair et al., 2005] is one such model that is inspired by a real-world sensor network coordination problem [Jain et al., 2009]. ND-POMDP assumes transition and observation independence and locality of interaction. Zhang and Lesser [Zhang and Lesser, 2011] used the ND-POMDP framework to model cooperative multi-agent 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. The interaction hypergraph is unchanged through the learning. In the NetRads domain, 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.

2.4 Multiagent Reinforcement Learning

MDPs are the foundation for much of the research in the single agent control learning. They provide a formal framework for modeling single-agent decision-making problems with uncertainty. Reinforcement learning (RL) allows an agent to learn the solution without knowing the details of a MDP. Reinforcement learning [Sutton and Barto, 1998] [Barto et al., 1989] [Whitehead and Ballard, 1991] is a mathematical framework used by agents to learn how to map situations to actions so as to maximize a numerical reward signal. The usual approach taken by reinforcement learning agents involves discovering which actions yield the most reward by trying them out, associating expected reward values with different agent states, and using reward values to choose actions.

When the transition function or reward function of an MDP is unknown, an agent can not directly compute the optimal policy and needs to learn it through interacting with the environment. Reinforcement learning is a field concerned with such learning in MDP environments. A broad spectrum of single-agent RL algorithms exists, e.g., model-free methods based on online estimation of value functions [Watkins and Dayan, 1992] [Peng and Williams, 1996] [Sutton, 1988] [Tesauro, 1992], and model-learning methods that estimate a model, and then learn using model-based techniques [Sutton, 1990] [Moore and Atkeson, 1993]. As in this research the multi-agent learning algorithms I describe and implement in the *Offline RL* (Figure 3.2) are built on top of Q-learning [Watkins and Dayan, 1992], I will briefly describe Q-learning here.

The Q-learning algorithm learns the optimal state-action value function. The state-action function (Q-value function) $Q^{\pi} : S \times A \rightarrow \Re$ defines the expected discounted reward of choosing a particular action from a particular state and then following the policy π. Formally, $Q^{\pi}(s,a) = \Sigma_{t=0}^{\infty} \gamma^t E r^{t+k} | s^k = s, a^k = a, \pi$. The optimal Q-value

function Q^* satisfies the Bellman optimality equation:

$$Q^*(s,a) = R(s,a) + \gamma \Sigma_{s' \in S} T(s'|s,a) max_{a' \in A} Q^*(s',a') \tag{2.3}$$

This equation states that the optimal value of taking a in u is the expected immediate reward plus the expected (discounted) optimal value attainable from the next state.

The policy is deterministic and picks the action with the highest Q-value for every state:

$$\pi(s) = \arg\max_{a \in A} Q(s,a) \tag{2.4}$$

The agent can achieve the learning goal by first computing Q^* and then choosing actions by the policy, which is optimal (i.e., maximizing the expected reward) when applied to Q^*.

Since the transition function is unknown, Q-learning turns Equation 2.3 into an iterative approximation procedure. The current estimate of Q^* is updated using estimated samples of the right-hand side of Equation 2.3. These samples are computed using actual experience interacting with the environment, in the form of the observed next state s^{k+1} and rewards r^{k+1} after taking action a^k in state s^k:

$$Q(s^k,a^k) \leftarrow Q(s^k,a^k) + \alpha^k [r^{k+1} + \gamma \max_{a^{k+1} \in A} Q(s^{k+1},a^{k+1}) - Q(s^k,a^k)] \tag{2.5}$$

Since its update rule does not require knowledge about the transition and reward function, Q-learning is model-free. The learning rate $\alpha^k \in (0,1]$ specifies how far the current estimate $Q(s^k,a^k)$ is adjusted towards the update target $r^{k+1} + \gamma \max_{a^{k+1} \in A} Q(s^{k+1}, a^{k+1})$. The learning rate is typically time-varying, decreasing with time. Separate learning rates may be used for each state-action pair. The expression inside the square brackets is the temporal difference, i.e., the difference between estimates of $Q(s^k,a^k)$ at two successive time steps, $k+1$ and k.

The sequence Q^k provably converges to Q^* under the following conditions [Watkins and Dayan, 1992] [Jaakkola et al., 1994]:

- Explicit, distinct values of the Q-function are stored and updated for each state-action pair.
- The time series of learning rates used for each state-action pair sums to infinity, whereas the sum of its squares is finite.
- The agent keeps trying all actions in all states with nonzero probability.

The third condition means that the agent must sometimes explore other actions than those determined by the current policy. To achieve this, one approach is to choose at each step a random action with probability . $\epsilon \in (0, 1)$ and the greedy action with probability $(1-\epsilon)$. This approach is called ϵ-greedy exploration. Another approach is to use the Boltzmann exploration strategy [Sutton and Barto, 1998], which in state s selects action a with probability:

$$\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \tag{2.6}$$

where $\tau > 0$ is the temperature that controls the randomness of the exploration. When $\tau \to 0$, this is equivalent with the policy specified by Equation 2.6. When $\tau \to \infty$, action selection is purely random. For $\tau \in (0, \infty)$, higher-valued actions have a greater chance of being selected than lower-valued ones.

This dissertation develops a decentralized learning paradigm to allow agents to effectively learn and adapt coordination policies in complex cooperative domains without explicitly building the complete decision models. *Multiagent Reinforcement Learning* (*MARL*) [Hu and Wellman, 1998] is a common approach for solving multi-agent 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. Most of the times each learning agent must keep track of the other learning (and therefore, non-stationary) agents.

Previous techniques of *MARL* have the problem of not converging in the worst case. Bowling and Veloso [Bowling and Veloso, 2002a] use a variable learning rate to

overcome this shortcoming. They present the *Win or Learn Fast heuristic* (WoLF) that makes a rational algorithm convergent in a two-agents, two-actions game. WoLF assumes that an agent knows the Nash Equilibrium and the strategy of the other players. For real-world applications like NetRads, this assumption does 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 [Abdallah and Lesser, 2006] is a variant of the WoLF [Bowling and Veloso, 2002b] 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. It helps determine how quickly or slowly an agent should change its policy while accounting for other agents that are learning. In my previous work [Cheng et al., 2010a] [Cheng et al., 2010b] [Cheng et al., 2010c], WPL is used to learn the meta-level control policies for the NetRads domain.

In [Zhang and Lesser, 2010], Zhang and Lesser experimentally showed that WPL converged slowly and did not perform well on some kinds of problems (e.g., Normal-form games, Shapley's game). Zhang and Lesser [Zhang and Lesser, 2010] 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.

Many AI researchers have addressed the use of abstraction for large-scale planning and problem solving. This is mainly because the complexity of states and actions in such large-scale domains makes policy convergence very difficult. Abstraction allows a system to focus on the information relevant to the decision making and ignore details that are irrelevant. In the RL literature, temporal abstraction [Sutton et al., 1999] and hierarchical control [Ghavamzadeh and Mahadevan, 2004] are used to combat

the curse of dimensionality in a principled way. One type of abstraction is the idea of a " macro-operator", or just a "macro", which is a sequence of operators or actions that can be invoked by name as if it were a primitive operator or action [McGovern and Barto, 2001]. Sutton et al. [Sutton et al., 1999] extended the usual notion of action in the framework of RL and MDPs 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. The theory of Semi-Markov Decision Processes (SMDPs) provides the foundation for the theory of options. McGovern et al. [McGovern and Barto, 2001] analyzed the roles of macro-actions that are closed-loop policies with termination conditions in RL algorithms. Empirical results showed that macro-actions may either accelerate or retard learning, depending on the appropriateness of the macro-actions to the particular task.

Ghavamzadeh & Mahadevan [Ghavamzadeh and Mahadevan, 2004] presented a hierarchical RL framework that builds upon the MAXQ framework [Dietterich, 2000] and the options model [Sutton et al., 1999]. Their approach studied how lower level policies over subtasks or primitive actions can themselves be composed into higher level polices. They showed that hierarchical RL using the MAXQ framework can be much faster and more compact than flat RL. They also showed that coordination skills are learned much more efficiently if the agents have a hierarchical representation of the task structure. These works emphasize the importance and advantages of abstraction in RL. My work in meta-level control is different from these works. I 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/macro-actions is inappropriate for the NetRads domain, since the environment is highly dynamic and a policy needs to be applied at the most 3 heartbeats to closely track the emerging weather phenomena.

In [Guestrin et al., 2001], the authors represented the multiagent system as a MDP

and showed how such factored value functions allow the agents to find a globally optimal joint action using a very natural message passing scheme. They used one single MDP to learn the policies the size of which blows up exponentially when the number of agents scales up. I use smaller MDPs to learn local policies, then coordinate them using heuristic rules to resolve conflicts. Following [Guestrin et al., 2001], Guestrin et al. [Guestrin et al., 2002] presented new algorithms for multiagent reinforcement learning that have the common features of parameterized, structured representation of a policy or value function. They proposed *coordinated reinforcement learning*, by which agents coordinate action selection activities as well as parameter updates. Their algorithms differ from the MAS learning algorithm used in NetRads in that structured communication and coordination between agents exists at both the learning algorithm and the execution stage.

Jelle R. Kok & Nikos Vlassis [Kok and Vlassis, 2006] described several algorithms for learning the behavior of a group of agents in a collaborative multiagent setting. They used the framework of coordination graphs of Guestrin et al. [Guestrin et al., 2001] which exploits the dependencies between agents to decompose the global payoff function into a sum of local terms. In this paper, the authors were interested in collaborative multiagent systems in which the agents have to work together to optimize a shared performance measure. This approach differs from other multiagent models, for instance, multiagent MDPs [Boutilier, 1999] or decentralized MDPs [Bernstein et al., 2002], in which all agents observe the global reward. In this approach, it assumes that the global reward is the sum of all individual rewards; while in NetRads, the global reward is not a simple function of all individual rewards (scanning pinpointing tasks between agents brings extra rewards which makes it complicated). Their model assumes that the environment is stationary, that is, the reward and transition probabilities are independent of the time step t. However, in my work, the environment is dynamic, the reward changes according to the time steps (e.g., new phenomena

moves in; old phenomena fades out; phenomena move fast between radars/MCCs).

2.5 MDP Unrolling

Agents often have large state spaces when they need to plan and coordinate in uncertain, dynamic environments. This makes decentralized learning computationally challenging.

Barto et al. [Barto et al., 1995] present a realtime dynamic programming algorithm that performs successive training cases on the uncertain environments. In each training case, value updates are only performed on the states actually visited in that case. Dean et al. [Dean et al., 1995] propose an envelope algorithm, which gradually extends the initial state space to add in more states and thus learn new policies. Given more computational time, this algorithm will perform better on partial policy calculation. These heuristic search algorithms increase effort spent on states that are most likely to be reached if an optimal policy is followed.

Wu and Durfee [Wu and Durfee, 2007] present a solver that learns effective policies in problems with large state spaces by selectively unrolling the search space. In their work, the exploration of the state space that is likely to be reached by the optimal policy is emphasized. In my work described here, I unroll the state space of each agent based on the conflict resolution performance it has achieved. Additionally, the policy in their work is determined based on the quality function that is known as a prior. I use a reinforcement learning algorithm to learn policies and a set of heuristics to guide the MDP unrolling. They execute the MDP unrolling at the deliberative level while I execute it as part of on-line learning.

Alexander et al. [Alexander et al., 2008] implement a meta-level control scheme to determine when the agent should stop unrolling the state space in order to derive a partial policy while reducing the time for re-prioritizing the states. Their approach collects performance profile information to make meta-level decisions on state expansion although I re-prioritize the set of heuristics that guide the unrolling using

actual on-line performance. In [Alexander et al., 2008], the MDP of a single agent is unrolled; while I am working on multiagent settings and unrolling MDPs of each agent.

2.6 DCOPs and Distributed Algorithms

In this dissertation, I use distributed constraint optimization algorithm to resolve conflicts from a global perspective for scenarios where negotiation mechanisms with local views are ineffective. Distributed constraint optimization problems (DCOPs) are a broad family of problems that can be used to model many domains including: optimal process control, task allocation and scheduling problems [Stranjak et al., 2008] and distributed sensor network management [Kho et al., 2009]. In a constraint satisfaction problem, the goal is to find a configuration of the domains of the variables so that they satisfy a set of constraints. A constraint optimization problem consists of a utility function that aggregates the payoffs for satisfying each of a set of 'soft' constraints over the states of variables in the problem [Schiex et al., 1995]. A distributed constraint optimization problem arises when a group of agents control the state of the variables in the system, having the purpose to maximize the global reward for satisfying all the constraints. A DCOP with n variables and m constraints consists of the tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{U} \rangle$ [Petcu and Faltings, 2005a], where:

- $\mathcal{X} = \{x_1, ..., x_n\}$ is a set of variables, each one assigned to a unique agent
- $\mathcal{D} = \{d_1, ..., d_n\}$ is a set of finite domains for each variable
- $\mathcal{U} = \{u_1, ..., u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility for each combination of values among these variables

An optimal solution to a DCOP instance involves an assignment of values in $\mathcal{D}$ to $\mathcal{X}$ such that the sum of utilities in $\mathcal{U}$ is maximal. Problem domains that require minimum cost instead of maximum utility view costs as negative utilities. The utility functions represent soft constraints but can also represent hard constraints by using

arbitrarily large negative values.

A number of complete algorithms that generate optimal solutions have been proposed for solving DCOPs: including OptAPO [Mailler and Lesser, 2004], ADOPT [Modi et al., 2005] and DPOP [Petcu and Faltings, 2005a] and variations of DPOP such as SDPOP [Petcu and Faltings, 2005b], M-DPOP [Petcu et al., 2008] and PC-DPOP [Petcu et al., 2007]. More specifically, OptAPO uses a partially centralized approach in which mediator agents compute solutions for portions of the overall problem. In contrast, ADOPT, DPOP and variations of DPOP preprocess the constraint graph, arranging it into a Depth First Search (DFS) tree, and then exchange messages over this tree.

Now, while these algorithms represent significant contributions in their own domain, they do not address many of the additional challenges that are present when the agents correspond to embedded devices. In particular, optimality demands that some aspect of these algorithms is exponential. For example, within OptAPO, mediator agents may be required to perform calculations that grow exponentially with the size of the portion of the overall problem that they are responsible for. Similarly, the number of messages that agents exchange when using ADOPT is exponential in the height of the DFS tree, and for DPOP, it is exponential in the width of the tree. Such exponential relationships are simply unacceptable for agent systems that exhibit constrained computation, bandwidth and memory resources.

In contrast, a large number of approximate stochastic algorithms have also been proposed for solving DCOPs. These algorithms are typically based upon entirely local computation, whereby each agent updates its state based only on the communicated (or observed) states of those local neighbors that influence its utility. As such, these approaches are well suited for large scale distributed applications, and in this context, the Distributed Stochastic Algorithm (DSA) [Zhang and Wittenburg, 2003] is one of the most promising; having been proposed for decentralized coordination

within sensor networks and bench-marked on DCOP problems [Zhang et al., 2005a]. However, algorithms of this type often converge to poor quality solutions since agents do not explicitly communicate their utility for being in any particular state, but only communicate their preferred state (i.e. the one that will maximize their own utility) based on the current preferred state of their neighbors.

Max-Sum is a decentralized coordination algorithm that provides approximate solutions for general constraint networks while requiring very limited communication overhead and computation. It uses message-passing and maximizes a utility function of a constraint graph. It is proven to work well for solving constraint optimization problems [Farinelli et al., 2008] in domains where approximate solutions are acceptable while requiring very limited communication overhead and computation. Empirical results show that Max-sum provides the approximate solution that are within 95% of the optimum [Stranders et al., 2009]. The Max-Sum algorithm defines a factor graph by creating a node for each variable and for each function. If the factor graph is cycle-free, the messages are guaranteed to converge [Farinelli et al., 2008], and the resulting solution will maximize the overall utility.

In the Max-Sum algorithm, there is a set of variables $x = \{x_1, x_2, ..., x_m\}$ on which a set of functions $F = \{F_1, F_2, ..., F_n\}$ depend. $F_i = F_i(x_i)$, $x_i \subset x$. The goal is to find x^* which satisfies the following:

$$x^* = \arg\max_x \sum_{i=1}^{n} F_i(X_i) \tag{2.7}$$

To achieve this, the Max-Sum algorithm defines a factor graph by creating a node for each variable and for each function. The graph is bipartite, and a function node is connected to a variable node if the corresponding function is dependent upon that variable. The algorithm has two types of messages passed between nodes, which are [Farinelli et al., 2008]:

Variable i to Function j:

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{k\in M_i\setminus j} r_{k\to i}(x_i) \tag{2.8}$$

Here α_{ij} is a scalar set such that $\Sigma_{x_i} q_{i\to j}(x_i) = 0$, and M_i contains the indices of function nodes connected to variable node i.

Function j to Variable i:

$$r_{j\to i}(x_i) = \max_{x_j\setminus i}[F_j(x_j) + \sum_{k\in N_j\setminus i} q_{k\to j}(x_k)] \tag{2.9}$$

where N_j contains the indices of variable nodes connected to function node j in the factor graph.

When the factor graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution. Furthermore, this convergence can be achieved in time equal to twice the depth of the tree by propagating messages from the leaf nodes of the tree to the root and back again [Farinelli et al., 2008]. In this case, the optimal variable assignment is found by locally calculating the function, $z_i(x_i)$, once the variable node has received a message from each of its connected function node.

$$z_i(x_i) = \sum_{j\in M_i} r_{j\to i}(x_i) \tag{2.10}$$

and hence finding $argmax_{x_i} z_i(x_i)$.

Otherwise, there is no guarantee of convergence. However, extensive empirical results show that, even in this case, the algorithm frequently provides good solutions. Before convergence, the value $z_i(x_i)$ of agent i calculated from incoming messages is actually an approximation of the exact value of action a_i given other agents act optimally. Therefore, the Max-sum algorithm can be implemented as an anytime algorithm by controlling the number of rounds of passing messages, which will trade off the quality and efficiency (or communication cost) of the action selection. In addition, the Max-sum algorithm is essentially distributed. Its messages are small

(linearly scaling with the maximum number of actions of agents), the number of messages typically varies linearly with the number of agents and hyperlinks, and its computational complexity scales exponentially with the maximum size of hyperlinks (which typically is much less than the total number of agents).

Mailler and Lesser [Mailler and Lesser, 2006] presented a complete, distributed algorithm for solving distributed constraint satisfaction problems (DCSPs). It is based on a cooperative mediation process that agents, when acting as a mediator, centralize small portions of the DCSP, and increase the size of their subproblems as the problem solving unfolds. My decentralized negotiation algorithm is similar in character to their approach, except that I introduce termination conditions to guarantee time bound for conflict resolution. Zhang et al. [Zhang et al., 2005b] described a technique that uses a partial-order schedule to enable agents to reason about the interactions among multiple negotiation issues. A contract bidding scheme is used to resolve conflicts among agents. There is no guarantee on solution quality and the time it runs mainly depends on the contracts bidding process. Krainin et al. [Krainin et al., 2007] presents a decentralized algorithm to solve the negotiation problem that arises when NetRads agents work together to improve the global performance. It uses marginal utility to compute better moves and updates neighborhood's configuration gradually until the pre-specified amount of time is reached. In my work, the agents learn the policies at the meta-level. To set a time limit for termination does not work for this problem, since it is difficult to pre-define the time limit which optimizes the performance of conflict resolution and reinforcement learning.

In my decentralized negotiation algorithm, agents are selected as mediators to resolve conflicts from the view of the neighborhood and the overall performance is improved by considering the importance as well as the number of different types of conflicts in the mediation process.

2.7 State of the Art Research on NetRads

There have been outcomes of other people with respect to the research of NetRads. Krainin et al. [Krainin et al., 2007] proposed a distributed task allocation mechanism in the NetRads system that improves the overall system utility with a significantly reduced computational load. Meta-level control was not introduced in their work and the optimization of radar scanning strategies was achieved at the deliberative level.

An et al. [An et al., 2011] introduced the concept "goal" to model end-users' preferences over multiple heartbeats and casts the complex sensing resource allocation problem as a continuous time optimization problem. In my simulation model, all the MCCs are myopically optimizing every "single" heartbeat's utility. 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. However, I use MARL algorithm to learn the meta-level policies for DEC-MDPs.

In [Kim et al., 2010], 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). In my work, I use MARL algorithm to learn meta-level policies for DEC-MDPs to maximize the utility of task scanning at each heartbeat.

CHAPTER 3: FORMAL FRAMEWORK

In this research, I mainly focus on cooperative MASs in which the agents have to optimize a shared performance measure. The definitions that are used in this research are described. The DEC-MDP framework for decentralized learning is introduced and the main contributions for constructing the framework are discussed. Abstractions of states and actions are used that decrease the exploration costs of DEC-MDPs efficiently. To address the complexity of DEC-MDP, the solution to the DEC-MDP is approximated by using a factored reward function to define the Nash Equilibrium instead of the global reward function.

3.1 Definition

I define the following terms that are used to describe the formal framework and my approach:

- Abstract State: An abstract representation of the state that captures the important qualitative state information relevant to the meta-level control decision making process. In this work, three features (will be discussed later in this Chapter) are used to construct an abstract state.
- Abstract Action: An abstract representation of the real domain-level action sets that capture the similar qualitative action information relevant to the meta-level control decision making process. In this work, the huge meta-level action space for radar reorganization is constructed using abstract actions.
- Detailed Action: An instantiation of an abstract action. When a certain agent takes an action, the "action" that is really implemented is a detailed action.
- 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 state for

agent i. Stochastic policies can cope with the uncertainty of observation and perform better than deterministic policies in partial observable environment.

- Learning Stage: An agent's offline learning process when it adapts its behavior to improve performance.
- Execution Stage: An agent's realtime execution process when it chooses and implements the appropriate policy.

3.2 A DEC-MDP based Approach

In real-world multiagent applications like NetRads, each agent has a partial view of the environment including a partial view of the other agents. In the NetRads domain, I consider a cooperative problem where all the MCCs share a common goal, the resolution is possible but intractable because of the complexity of the problem. Such team problems can be modeled by DEC-MDP [Bernstein et al., 2002]. The decentralized meta-level control in NetRads is framed as a stochastic, factored DEC-MDP which is a DEC-MDP where the policy for each agent can be stochastic. A DEC-MDP [Bernstein et al., 2002] 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 [Boutilier, 1999] and DEC-POMDP [Bernstein et al., 2002] concerns the observability assumption: MMDP uses full observation of the global state; DEC-POMDP uses only partial observation. The multiagent meta-level control problem (n agents in the system) is mapped to a factored DEC-MDP model in the following way. The model is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, 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 *abstract state* as defined in Chapter 3.2.1 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.

- $\mathcal{A} = A_1 \times A_2 \ldots \times A_n$ 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 *abstract action* that includes radar reorganization and heartbeat adaptation as defined in Chapter 3.2.2.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ 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. $R_i : S \times A \rightarrow \mathcal{R}$ 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 Chapter 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.
- $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ is a finite set of joint observations. Ω_i is the set of observations for agent i.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow \mathcal{R}$ is the observation function. $\mathcal{O}(s, (a_1 ... a_n), s', (o_1 ... o_n))$ is the probability of agents 1 through n seeing observations o_1 through o_n (agent i sees o_i) after the sequence $s, (a_1 ... a_n), s'$ occurs.
- Joint full observability: the $n-tuple$ of observations made by the agents together fully determine the current state. If $\mathcal{O}(s, (a_1 ... a_n), s', (o_1 ... o_n)) > 0$ then $\mathcal{P}(s' \mid (o_1 ... o_n)) = 1$.

NetRads is modeled as a DEC-MDP with the following characteristics (defined in Chapter 2.3.3):

- transition independent
- observation independent
- not locally fully observable

- not reward independent

In NetRads, multiple $MCCs$ are allowed to collect weather information from the same/overlapping area simultaneously without conflicting meaning the $MCCs$ do not interact through their state transitions. However in general the NetRads does not exhibit transition independence because a MCC_i's actions affect its radar configuration and the information it has collected, as well as other agents' radar configurations and the information they have collected. There are exceptions where DEC-MDPs representing the NetRads decision making are transition independent as shown in the following scenarios : 1) Each MCC only has its own local tasks. In other words, there are no shared tasks in the overlapping regions between $MCCs$. 2) $MCCs$ have both local tasks and shared tasks. However all the shared tasks in the overlapping regions are non-pinpointing tasks meaning that the agents can make independent decisions such that the new local state of each agent depends only on its previous local state, the action taken by that agent, and the current external features.

NetRads exhibits observation independence. MCC_i only observes its own local state and sees nothing related to any other $MCCs$.

NetRads is not locally fully observable. MCC_i's observation of its local state depends on its action as well as its neighboring MCCs' actions (Its neighboring MCCs may hand off radars to MCC_i that changes its radar configuration, so that changes its local state). So when MCC_i takes an action, it does not fully observe the outcome, which is MCC_i's next local state.

NetRads does not exhibit reward independence. The local reward of MCC_i represents the average of qualities of tasks in its neighborhood. It depends on the actions of multiple MCCs (scanning pinpointing tasks simultaneously would increase the reward value).

3.2.1 Abstract States

Extending the definitions of real state and state abstraction in Raja and Lesser [Raja and Lesser, 2007], I 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. This abstraction of the real state is called the *abstract state* [Cheng et al., 2010a].

Three features F_0, F_1 and F_2 are defined that sufficiently capture the meta-level state information critical to the meta-level decision making process as an abstract representation at each MCC. I use the motivating example introduced in Chapter 1 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 $V_i^{hb} \in \{30\ seconds, 60\ seco\ nds\}$ 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 Chapter 1 (Figure 1.7), MCC_2 has a 60 seconds heartbeat and has two radars (R_5 and R_7) involved in the data correlation with its neighboring MCCs. MCC_2 has the feature $F_0 = (60seconds, many)$ in its meta-level state.

Feature F_1 contains *Information about Neighbor(s).* It is the *Neighborhood Scenario*

NS_i defined earlier in Chapter 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 Chapter 1.1. Determining F_2 also involves communication between MCC_i and its neighbors.

In the Figure 1.7 example, MCC_2 has the initial state: s^0, in which $F_0 = (60seconds, many)$, $F_1 = \langle(60seconds, 1), (60seconds, 1), (60seconds, 1)\rangle$ and $F_2 = \langle Medium, Low, High\rangle$. Thus, the total number of meta-level states in this example is 55566 (F_0, F_1 and F_2 has 6, 343 and 27 domain values respectively; $6 \times 343 \times 27 = 55566$).

3.2.2 Abstract Actions

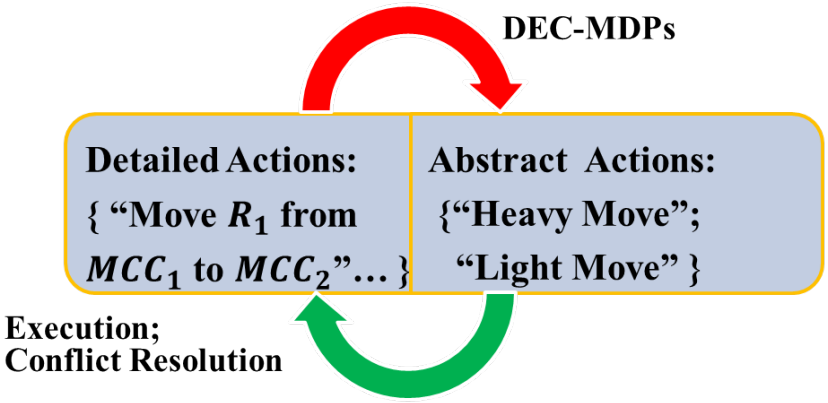


Figure 3.1: Relation between abstract action and detailed action.

As described in Chapter 1.1.2, 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 real actions are used, and when the action space blows up, the complexity of solving the associated DEC-MDP increases exponentially. The *abstract actions* (defined in Chapter 3.1) for radar reorganization include two qualitative modes. In this work, only two qualitative modes are used since the average number of radars each MCC controls is low. Setting more modes is prone to result in more redundant states. 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 abstract actions, for each neighbor of MCC_i, MCC_i has 3 possible choices {ϕ, *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. In the case that $x = 8$ and $y = 3$, using abstract actions reduces the number of explored states in the MDP by 99.9% ($(3 + 1)^8 = 65536$, $3^3 = 27$).

I now discuss the role of detailed actions. Suppose MCC_i has a 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 *abstract 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 Figure 1.7, one action for

MCC_2 could be "$LightMove(MCC_2\ to\ MCC_1)$ & $LightMove(MCC_2\ to\ MCC_3)$".

As Figure 3.1 shows, when a certain agent takes an action, the "action" that is really implemented is a detailed action (defined in Chapter 3.1). Each meta-level action for radar reorganization could have different detailed actions associated with it. For example, in Figure 1.7., "$HeavyMove(MCC_3\ to\ MCC_2)$"has the detailed actions such as: "Move R_9 R_{10} and R_{11} to MCC_2"and "Move R_9, R_{10}, R_{11} and R_{12} to MCC_2". The heuristic rule-based algorithm that changes the detailed actions of meta-level actions to resolve conflicts will be described in Chapter 4.

3.2.3 Reward Function

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 correctly represent 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 average of utilities (instead of sum of utilities) of the tasks is used to reflect the radar scanning performance. During the learning stage, each agent communicates with its neighbors to compute its reward from a neighborhood perspective. 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 tips the Nash equilibrium approximate solution of the DEC-MDP closer to the global reward which is actual solution to the stochastic DEC-MDP. Stochastic DEC-MDPs do not always have a well-defined no-

tion of optimal behavior for a particular agent, since its performance may depend on behaviors of other agents. The most common solution concept in these cases is Nash equilibria [Osborne and Rubinstein, 1994] [Singh et al., 2000], intuitively defined as a particular behavior for all the agents where each agent is acting optimally with respect to the other agents' behaviors.

3.3 Control Flow

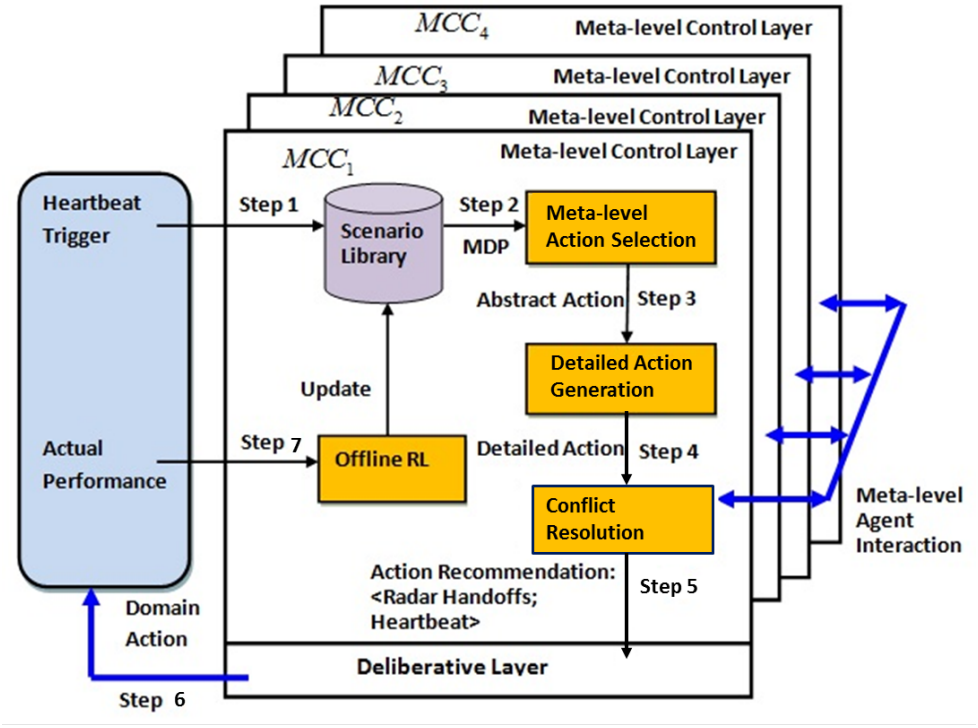


Figure 3.2: Control flow of each MCC involving 4 MCCs.

Figure 3.2 describes the control flow of my learning approach within each agent. Each MCC has a meta-level control layer that includes five sub-stages:

- The Offline RL: It learns the policies for each weather scenario offline by extending PGA-APP (mentioned in Chapter 1).
- The Scenario Library: It is a centralized module that stores the MDPs of each weather scenario as well as the meta-level policies for each abstract state.
- The Meta-level Action Selection: It chooses the appropriate abstract actions based on the policies.

- The Detailed Action Generation: It maps the abstract actions to the detailed actions associated which include radar/MCC reconfiguration and heartbeat adaptation. The mappings of these meta-level actions to detailed action sets are stored in this component.
- The Conflict Resolution: It resolves conflicts that are resulted among the agents' detailed actions.

In the next Chapter, I will describe the functionality and interaction of each component in more details.

3.4 Summary

In NetRads, the agents are cooperative and are in a dynamic environment which is partially observable. The new state and received reward for one agent also depend on the actions selected by the other agents. The communication among agents needs to be controlled at a reasonable level so that the overall performance is optimized. The DEC-MDP framework is presented which approximates the DEC-POMDP to learn the policy for each agent. Abstract states and abstract actions are constructed where instances within a class have similar features to handle the complexity of the real state and action space. The MDP search space is decreased substantially by using abstract states and actions. The agents learn stochastic policies and approximate the solution to the DEC-MDP by using a factored reward function to define the Nash equilibrium. In the next chapter, I will describe local learning based on my DEC-MDP framework, present one multiagent reinforcement learning algorithm that learns local policies for radar reorganization and heartbeat adaptation, and discuss about the empirical results.

CHAPTER 4: LOCAL LEARNING

In this chapter, I address the following questions: Can agents automatically learn policies for specific environments based on the DEC-MDP framework described in Chapter 3? Does this learned policy perform well for that environment and improve the system performance?

As discussed in Chapter 3, this research focuses on cooperative agents that maximize the social utility by successfully completing their individual goals in the context of limited computation and absence of detailed models of the environments. Reinforcement learning is useful for agents to learn local policies in such contexts. In Chapter 4.1, the complexities of the issues faced by multiagent reinforcement learning agents are described and the state-of-the-art algorithm that I extend in the context of my DEC-MDP framework for local policy learning is presented. In Chapter 4.2, the control flow that is designed for agents' policy learning in simplified environments is described. Experimental results describing the performance of the learned policies using my local learning approach are provided.

4.1 Policy Gradient Ascent with Approximate Policy Prediction (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-stationarity of environments, like NetRads, where multiple interacting agents are learning simultaneously, single-agent reinforcement learning techniques are not guaranteed to converge in multiagent settings. The central challenge for multi-agent learning is that each learner is adapting its behaviors in the context of other co-adapting

learners. When applying single-agent learning to stationary environments (e.g., MDP problems), the agent experiments with different policies by interacting with the environment until discovering a globally optimal policy. In dynamic environments, the agent may at best try to keep up with the changes in the environment and constantly track the shifting optimal behavior.

In most MARL algorithms, a common assumption 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 [Banerjee and Peng, 2007] [Bowling and Veloso, 2002a] [Conitzer and Sandholm, 2007], or observe what actions other agents executed and what rewards they received [Conitzer and Sandholm, 2007] [Hu and Wellman, 2003]. For practical applications like NetRads, 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. Thus it is important for agents to learn local policies that leverage predictions about other agents' actions.

Zhang and Lesser [Zhang and Lesser, 2010] 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.

I augment the PGA-APP algorithm to the domain of NetRads to learn the MMLC policies offline. 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 *abstract actions* and the states are *abstract states* as defined in Chapter 3.2. The $\epsilon-Greedy$ [Sutton and Barto,

1998] exploration scheme is used to pick actions for learning (line 4, Algorithm 1). The $\epsilon - Greedy$ exploration scheme is the best known scheme to balance the exploration with exploitation by selecting both greedy actions and random actions during the learning stage.

Algorithm 1 Zhang & Lesser's PGA-APP Algorithm

1: Let θ and η be the learning rates, ξ be the discount factor, γ be the derivative prediction length;
2: Initialize value function Q and policy π;
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*;

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 Chapter 3.2.3. I introduce 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, this immediate reward does not need to be computed, and hence avoid any extra communication among MCC agents. The communication overhead 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. I also introduce communication among MCC's neighbors in the learning and execution stages to calculate some features of the abstract 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 [Zhang and Lesser, 2010]. 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. In the next Chapter, we describe how we extend PGA-APP in our learning framework to learn offline policies.

4.2 Offline Learning in Simplified Environment

As described in Chapter 3.3, Figure 3.2 shows the control flow within each MCC. The *Scenario Library* component is a centralized module that stores the MDPs of each weather scenario as well as the meta-level policies for each abstract state. The agents are assumed to operate in a *simplified environment* that the entire radar network sees only one specific scenario at any point in time. It is also worthwhile to note that I do not include as a component in the abstract state having local neighborhood weather pattern. In this work, each MCC is assumed to know the class of weather scenario for the whole system by checking the number of each *task* pre-defined in the training/test case. Consider the motivating example in Chapter 1, suppose the weather scenario is HRLS, each MCC chooses the MDP for HRLS and applies its policy according to its current abstract state (Steps 1 and 2 in Figure 3.2). The policies for each weather scenario are learned offline using PGA-APP which is the role of the *Offline RL*. The *Meta-level Action Selection* component chooses the appropriate abstract action based on current policy, depending on the specific weather scenario that the agent is in. For example, MCC_2 chooses its abstract action ("Light Move", "Use shorter heartbeat") based on using the MDP policies for HRLS (Step 3 in Figure 3.2).

The *Detailed Action Generation* component maps the abstract actions to the associated detailed actions which include radar/MCC reconfiguration and heartbeat

adaptation. This sub-stage uses a greedy search to compute the first detailed action that meets the condition of "Light Move"or "Heavy Move". This may cause the conflicts among detailed actions of the abstract actions of neighboring MCCs and this is the main reason that I introduce the idea of conflict resolution and coordinating decentralized learning and conflict resolution to converge to optimal policies. In Chapter 5, I will describe the conflicts that can arise between locally optimal agent policies and present my incremental approach to resolve conflicts and guide the decentralized learning process. Suppose MCC_2 chooses the detailed action of its meta-level action: "Move R_4 to MCC_3" and "Use 30 seconds heartbeat"(Step 4 in Figure 3.2). As a result of the *Conflict Resolution* component, MCC_2 changes its detailed action to "Move R_5 to MCC_3" and "Use 30 seconds heartbeat"(Step 5 in Figure 3.2). At runtime, when the *MMLC* phase is triggered every heartbeat, each MCC agent adopts the scenario-appropriate policy and executes the detailed action of its meta-level action. The feedback from the *Offline RL* component updates the MDPs in the *Scenario Library*.

4.2.1 Scenario Library

In this section, I describe how the *Scenario Library* component is constructed. As mentioned in Chapter 1, it is important that the *MMLC* phase takes negligible amount of time so that there is enough time for the complex operations of *Local Optimization* and *Negotiation* phases. Online learning on a very large MDP that captures all detailed weather scenarios, states and actions during the *MMLC* phase can be very time expensive. In DEC-MDPs for a subset of agents, because the policy space can not be independently searched, the size of policy space blows up very quickly even when the number of agents are limited. Also, learning such a DEC-MDP is very difficult because each possible action/state space combination has to be visited many time during learning.

The *Scenario Library* that stores the MDPs of each weather scenario as well as the

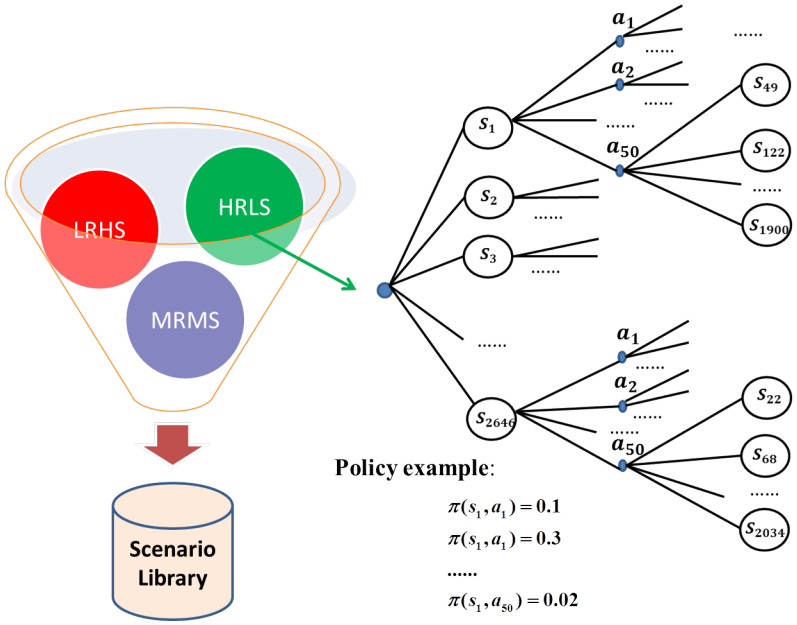


Figure 4.1: Construction of the Scenario Library.

meta-level policies for each abstract state is constructed. I argue that the learning is sped up by categorizing different weather scenarios and learning policies for each MDP of each weather scenario. In this work, 9 MDPs for different weather scenarios are stored in the *Scenario Library* and three of them (HRLS, LRHS and MRMS) are used to do the evaluation. For each MDP that is stored in it, there is a policy associated with each abstract state. As discussed in Chapter 1, the policies that are learned and applied are stochastic policies. Since an agent has no direct access to the current state in DEC-MDPs, selecting actions based on the current state (as in a MDP) is no longer valid. Stochastic policies can cope with the uncertainty of observations to a certain degree and perform better than deterministic policies in partial observable environments.

Figure 4.1 shows what is stored in the *Scenario Library*. In this example, I consider the NetRads system with 3 MCCs and 18 radars. The *Scenario Library* stores three MDPs as well as the policies for each abstract state. For the MDP of HRLS, the total number of abstract states is 2646 (F_0, F_1 and F_2 has 6, 49 and 9 domain values

respectively; $6 \times 49 \times 9 = 2646$). The total number of abstract actions is 50 (25 domain values for radar reorganization and 2 domain values for heartbeat adaptation). S_1, $S_2 ... S_{2646}$ are the abstract states; a_1, $a_2 ...$ a_{50} are the abstract actions.

Figure 4.2 shows the dominant abstract actions of MCC_1 under different abstract states for HRLS which is stored in the *Scenario Library*. The *dominant abstract action* under abstract state s for MCC_i is defined as the abstract action that has the highest probability distribution in the policy $\pi_i(s)$. For example, suppose $\pi_i(s)$: $a_1(30\%); a_2(50\%); a_3(20\%)$, the dominant abstract action prescribed by $\pi_i(s)$ for state s is a_2. F_0, F_1, and F_2 are the features defined in Chapter 3.2.1. In this example, the total number of policies of MCC_1 in HRLS is 2646.

F_0	F_1	F_2	Dominant abstract action
$(HRLS, 60seconds, 1)$	$\langle(60seconds, many), (60seconds, 1), (60seconds, 1)\rangle$	$\langle High, Low, Low\rangle$	'Light Move(MCC_1 to MCC_2)'; 'Use 60 seconds heartbeat'
$(HRLS, 60seconds, many)$	$\langle(60seconds, many), (60seconds, 1), (60seconds, 1)\rangle$	$\langle Medium, Low, Low\rangle$	'Heavy Move(MCC_1 to MCC_2)'; 'Use 60 seconds heartbeat'
$(HRLS, 60seconds, many)$	$\langle(30seconds, many), (30seconds, 1), (30seconds, 1)\rangle$	$\langle High, Low, Low\rangle$	'Heavy Move(MCC_1 to MCC_2)'; 'Use 30 seconds heartbeat'
$(HRLS, 60seconds, 1)$	$\langle(60seconds, 1), (60seconds, many), (60seconds, many)\rangle$	$\langle High, Low, Medium\rangle$	'Light Move(MCC_1 to MCC_4)'; 'Use 60 seconds heartbeat'
$(HRLS, 60seconds, many)$	$\langle(60seconds, 1), (60seconds, many), (60seconds, 1)\rangle$	$\langle Low, High, Low\rangle$	'Heavy Move(MCC_1 to MCC_3)'; 'Use 60 seconds heartbeat'
.....	.....	.....	.....
$(HRLS, 30seconds, many)$	$\langle(30seconds, 1), (60seconds, 0), (60seconds, 1)\rangle$	$\langle Medium, Low, High\rangle$	'Light Move(MCC_1 to MCC_2)'; 'Light Move(MCC_1 to MCC_4)'; 'Use 60 seconds heartbeat'
.....	.....	.....	.....

Figure 4.2: Dominant abstract actions of MCC_1 under different states for HRLS which is stored in the *Scenario Library*.

4.2.2 Offline RL

The policies for each weather scenario are learned offline using PGA-APP which is the role of the *Offline RL* component (Figure 3.2). 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, during the learning phase a simplifying assumption is made in my 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, communication in offline learning is controlled since my reward function is targeted for policy generation and not necessarily for execution.

During the learning stage, the *MMLC* phase is triggered at every heartbeat which means the meta-level actions of the MCCs are changed at every heartbeat. This is acceptable in the NetRads environment, 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. My evaluation results show that triggering *MMLC* at every heartbeat during learning helps improve the overall performance. It is empirically observed that if MMLC is triggered multiple heartbeats apart during learning, it will result in a meta-level policy, that is obsolete due to dynamic nature of the environment. In other words, the weather phenomena are changing quickly and dynamically resulting in significant changes of the current meta-level states of the MCCs.

During the execution stage, the *MMLC* phase 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 action based on its meta-level state. 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 5 in Figure 3.2).

In the next Section, the ability of agents to learn local policies in a decentralized fashion is evaluated for a range of weather scenarios. Our learning algorithm is compared with others to show meta-level control is useful and our algorithm allows the network of NetRads to dynamically adjust to changing weather phenomena.

4.3 Experiments

I use the simulator of the NetRads radar system [Krainin et al., 2007] to evaluate my implemented system. 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*.

4.3.1 Experiment Setup

For the experiments reported here, I use the simulation setup where there are 3, 12 and 30 MCCs (agents). This is the setup used by Krainin et. al [Krainin et al., 2007]. The experiments are run on a single machine although it is assumed that there are several computation units working in parallel in a time step simulation. Figure 4.3 is the snapshot of the radar simulator for a particular real-time scenario. In Figure 4.3, each hollow circle represents a radar and each filled circle represents a task (*rotation* and *storm* tasks are only concerned about in the evaluation.). The Radar Information Panel (Figure 4.3) 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*. I test the results for three different

types of *weather scenarios*: HRLS, LRHS, and MRMS. There are 80 total tasks in each *weather scenario*. HRLS contains 60 rotation tasks, 20 storm tasks as well as each of the other two types; LRHS contains 60 storm tasks, 20 rotation tasks as well as each of the other two types; MRMS contains 40 storm tasks, 40 rotation tasks as well as each of the other two types.

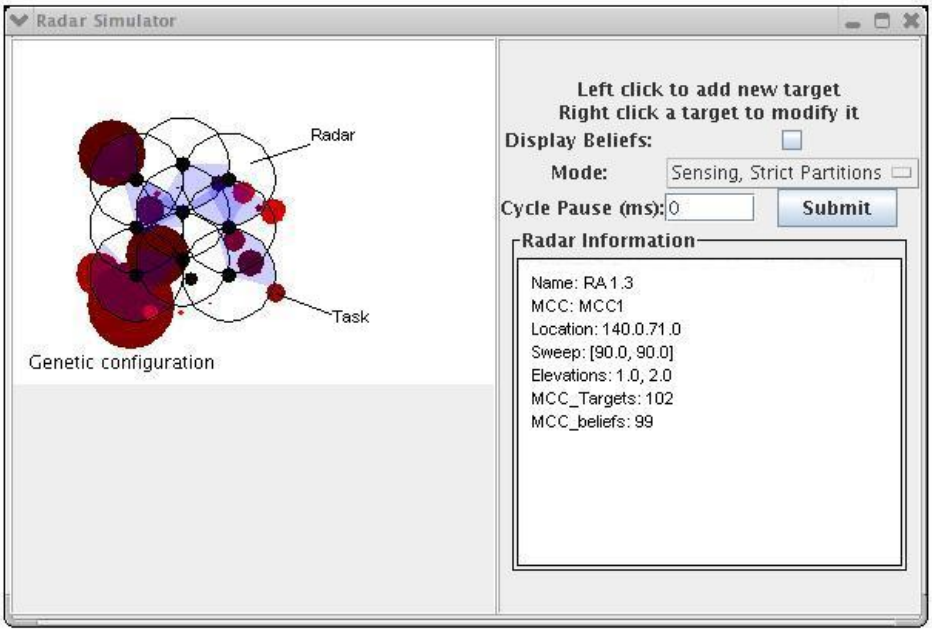


Figure 4.3: Snapshot of Radar Simulator.

I 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. *percentPinpointing* is varied to evaluate the performance on different numbers of pinpointing tasks. The number of tasks in training/test cases is also scaled up. *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 [Kurose et al., 2006]. 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, I make 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.

The results of three algorithms are compared: *No-MLC*, *Adaptive Heuristic Heartbeat (AHH)* and *PGA-APP*.

- *No-MLC* is the algorithm with no explicit or implicit meta-level control (It has all the phases except *MMLC* in a heartbeat).
- *AHH* is the algorithm where I 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 handoff issues. Assigning the same heartbeat to the neighboring MCCs with overlapping region results in better communication/negotiation in the *Negotiation* phase so as 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.

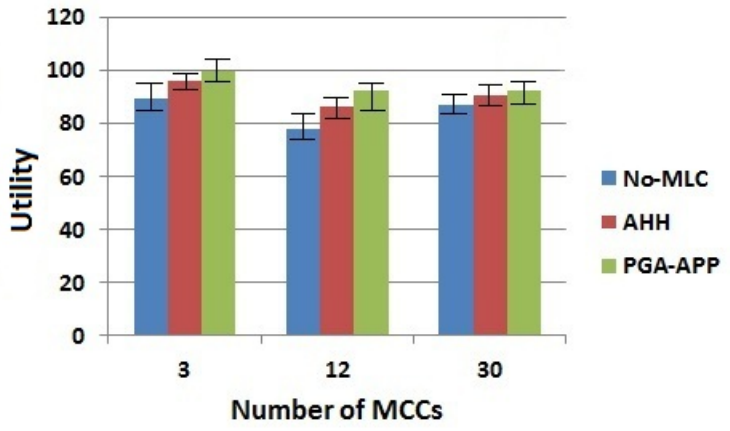
For the *MMLC* phase, I used 1000 training cases and each has a long sequence of training data to learn the policies for each abstract scenario offline.

Learning parameters (defined in Line 1, Algorithm 1) will affect the convergence of PGA-APP. For non-competitive problems (e.g., NetRads), with a too large 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 a 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 [Zhang and Lesser, 2010]. 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. I ran 30 test cases for each of the three algorithms described above for the three different weather scenarios (HRLS, LRHS and MRMS).

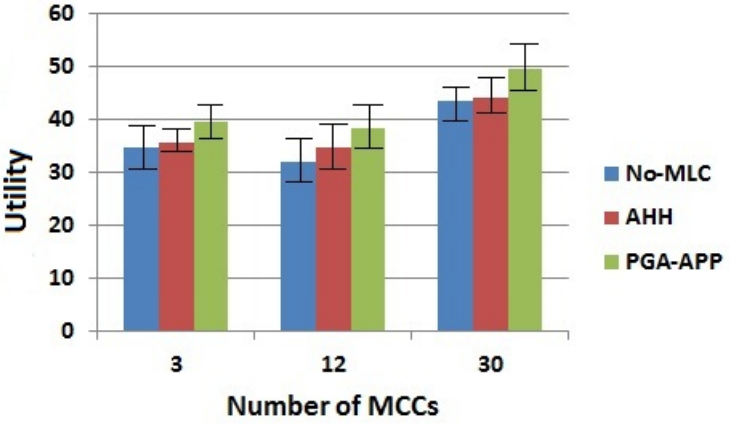
In the experimental evaluation, PGA-APP *is compared with* No-MLC *and* AHH. *Results show that adaptive multiagent meta-level control significantly improves the performance for a variety of scenarios.*

4.3.2 Performance of *PGA-APP*

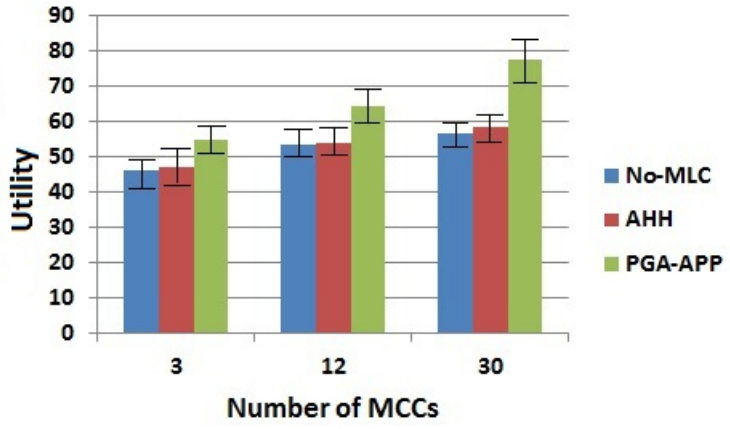
I ran test cases for each weather scenario with 3, 12 and 30 MCCs (*percentPinpointing* is set to 60%, the number of tasks is 80. Each MCC controls 5 radars initially.). Figure 4.4 shows the performance of *No-MLC*, *AHH* and *PGA-APP* on *Utility* for a variety of scenarios. In HRLS scenarios, all the MCCs have to handle HRLS scenarios simultaneously. *AHH* performs significantly ($p < 0.05$) better than *No-MLC* on *Utility* in all comparisons (Figure 4.4(a)). This shows the effectiveness



(a) **HRLS Scenarios**



(b) **LRHS Scenarios**



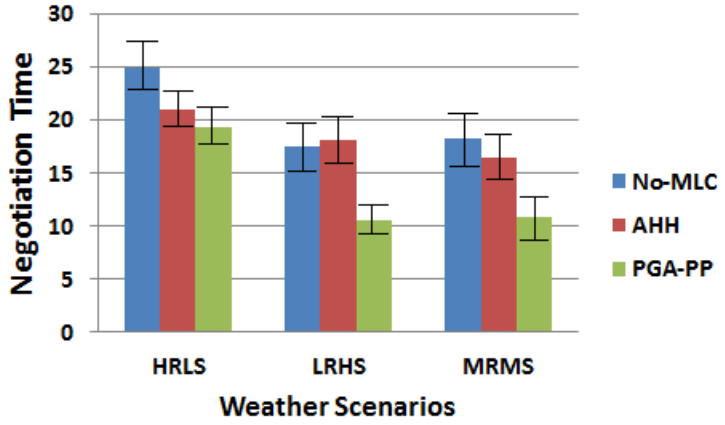
(c) **MRMS Scenarios**

Figure 4.4: *Utility* of *No-MLC*, *AHH* and *PGA-APP* in different weather scenarios for number of MCCs to be 3, 12 and 30.

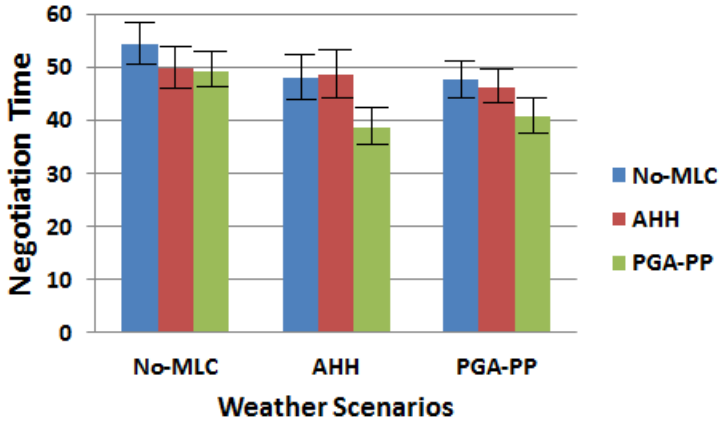
of adding meta-level control to agent reasoning in HRLS scenarios. According to the hand-generated rules in *AHH*, the three MCCs would all set their heartbeat to 60 seconds for HRLS. The three MCCs would then have more time on *Local Optimization* and *Negotiation* so that the final configurations of scanning tasks for the next heartbeat would be more optimized. This results in larger *Utility*. In HRLS scenarios, *PGA-APP* performs significantly (p values from t-tests are 0.0074, 0.037 and 0.016 respectively) better than *No-MLC* and a little better than *AHH*. The minor discrepancy of performance between *PGA-APP* and *AHH* on HRLS scenarios leads to the speculation that the 60 seconds heartbeat is critical for rotations due to the need for more scanned elevations. Rotations need more time for scanning as they must be scanned at the lowest six elevations. Storms, on the other hand, must be scanned at the lowest four elevations to obtain useful information.

In both LRHS and MRMS scenarios (Figure 4.4(b) and Figure 4.4(c)), *AHH* performs a little better than *No-MLC*. *PGA-APP* performs significantly better than *No-MLC* (In LRHS scenarios, p values are 0.023, 0.0095 and 0.0071 respectively; In MRMS scenarios, p values are 0.008, 0.029 and 0.035 respectively) and *AHH* (In LRHS scenarios, p values are 0.0086, 0.0013 and 0.0028 respectively; In MRMS scenarios, p values are 0.0074, 0.0082 and 0.034 respectively). It is observed that the 30 seconds heartbeat is not a profound factor in LRHS scenarios (*AHH* increases small amount of *Utility*.). In *PGA-APP*, each MCC adopts the policy appropriate to its *neighborhood scenario*. Allocating radars with large data correlation to the same MCC reduces the time for negotiation between MCCs which would increase the time for *Local Optimization*. In certain situations (e.g., there are many internal tasks compared to boundary tasks) it is better to do a good job on 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). To summarize, *PGA-APP* performs significantly better on learning policies

to control when and which radars should be moved.



(a) **30 seconds heartbeat**



(b) **60 seconds heartbeat**

Figure 4.5: *Negotiation Time* of *No-MLC*, *AHH* and *PGA-APP* in different weather scenarios.

In Figure 4.5(a) and Figure 4.5(b), *PGA-APP* performs significantly better than *No-MLC* on *Negotiation Time* (p values are 0.0028, 0.033 and 0.0058 respectively) for each weather scenario. *PGA-APP* uses least time on *Negotiation* phase and achieves highest *Utility* in each weather scenario. This shows that adaptive meta-level 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. *AHH* does not perform better than *No-MLC* on all weather scenarios (It spends more *Negotiation Time* than *No-MLC* in LRHS scenarios) since

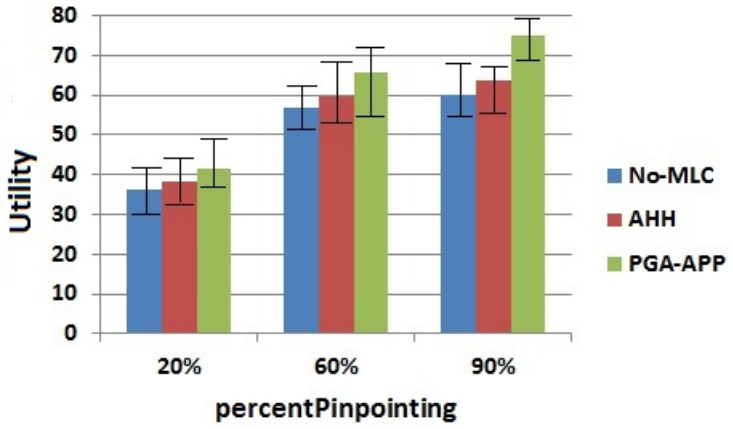


Figure 4.6: *Utility* of *No-MLC*, *AHH* and *PGA-APP*, for *percentPinpointing* to be 20%, 60% and 90%.

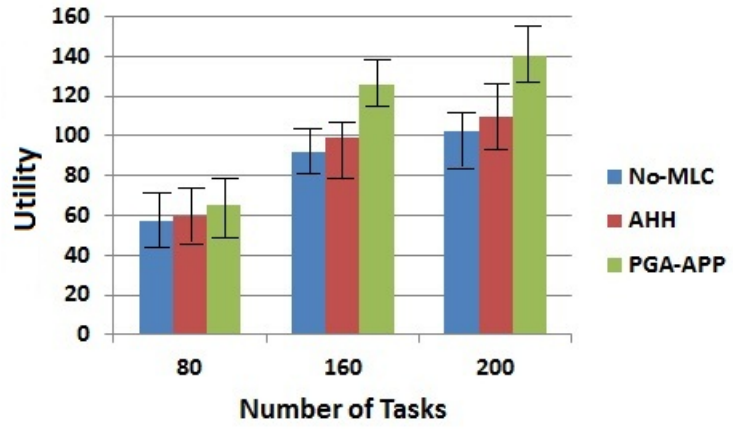


Figure 4.7: *Utility* of *No-MLC*, *AHH* and *PGA-APP*, for number of tasks to be 80, 160 and 200.

AHH is not as adaptive as *PGA-APP* in dynamic conditions.

percentPinpointing (setting it to 20%, 60% and 90%) was varied and test cases were run on all the three weather scenarios. In Figure 4.6, it is observed 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 so as to increase *Utility* of all the scanning tasks. In all *percentPinpointing* settings (20%, 60% and 90%), *AHH* performs better than *No-MLC* and *PGA-APP* achieves

the best performance.

In Figure 4.7, I 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. *PGA-APP* performs significantly better than *No-MLC* (p values are 0.0046, 0.023 and 0.0076 respectively) and *AHH* (p values are 0.049, 0.00063 and 0.0035 respectively) on *Utility*.

The results of my evaluation show that PGA-APP performs significantly better on learning meta-level policies to control radar reorganization and heartbeat adaptation.

4.4 Summary

This chapter describes a multiagent reinforcement learning approach which equips agents to automatically learn local policies. This approach allows agents to adapt to the dynamics of other agents and the environment and improves the system performance for a cooperative MAS. This approach learns offline policies with a simplifying assumption that the entire network experiences one particular weather scenario. The learning is sped up by categorizing different weather scenarios and learning policies for each MDP of each weather scenario. At the execution stage, each agent chooses the scenario-appropriate MDP and implements its policy. The utility of this approach is demonstrated experimentally by showing that the policies learned by the agent performs significantly better than the hand-generated heuristic policies.

In the next chapter, I will discuss about my approach and handling conflicts that may occur between agents' local policies, especially when agents experience complex environments. Three incremental approaches will be presented that: a) resolve conflicts locally using heuristic rules; b) resolve conflicts from a partially global perspective and c) resolve conflicts from a global perspective. Experiments comparing and describing the viabilities of these approaches in different system environments are presented.

CHAPTER 5: CONFLICT RESOLUTION

As described in Chapter 4, the policies learned using the PGA-APP algorithm for each MDP could be optimal policies for each agent from a local perspective. However, it is possible that in some environments the chosen abstract actions could cause conflicts between agents when the associated detailed actions are generated. As will be discussed in the experimental section, resolving such conflicts in an intelligent manner improves overall system performance. Chapter 5.1 describes the types of conflicts. Chapter 5.2 describes a heuristic rule-based algorithm that uses pre-defined rules to locally resolve the conflicts by changing the detailed actions of the abstract actions among agents at both learning and execution stages. This approach is capable of resolving local conflicts when there are few agents and the network is not highly constrained. In the more uncertain and complicated environments that lead to very huge search space, decentralized online policy learning is computationally challenging. Chapter 5.3 describes a decentralized learning algorithm that uses policies learned in a simple environment in order to smartly expand the search space in the real environment, when conflicts arise. A decentralized negotiation algorithm is presented to resolve conflicts from a partially global perspective and hence guide the state expansion and policy learning process. Empirical results show that the learning algorithm achieves good performance in the complicated and dynamic environments. Chapter 5.4 emphasizes on how I apply a state of art DCOP algorithm, Max-sum [Farinelli et al., 2008], in the learning approach to resolve conflicts in a global perspective and learn globally optimal policies. Experimental results comparing these approaches in different scenarios are analyzed and summarized.

5.1 Conflict Type

The concept of a conflict in this work is defined as incompatibilities among two or more agents' local policies for such reasons that agents compete for the same shared resource, agents fail to balance the load of the whole multiagent system and agents are not synchronized in communication. For the NetRads domain, I identify the following types of conflicts among agents' detailed actions associated with abstract actions:

(a) *Local Radar Conflicts (LRC)* refer to situations in which an agent receives radars from two or more neighboring agents simultaneously and thus is overloaded. Consider the situation (Figure 1.7) 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 would result in a very high load for MCC_3 (Control of too many radars could increase a MCC's time and messages for local negotiation). The threshold for a high load of the MCC varies according to different simulation scenarios. 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 actions. If MCC_i finds that its neighbor MCC_j has the detailed action 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 Figure 1.7, 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 actions. If MCC_i finds that its neighbor MCC_j has the detailed action 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 during the *Negotiation* phase. The MCCs are assumed to communicate with neighboring MCCs that have the same heartbeats during the *Negotiation* phase. Suppose in Figure 1.7, 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 Chapter 1.1. An IHC is recognized when MCC_i finds the neighboring agent MCC_j tries to use a different heartbeat by exchanging messages.

5.2 The Heuristic Rule-based Algorithm

5.2.1 Algorithm Description

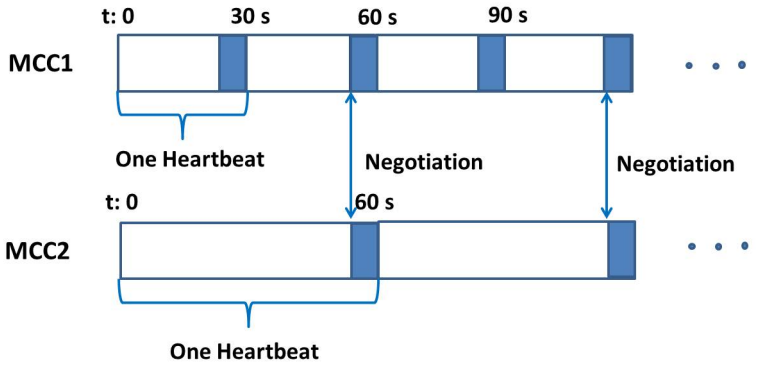


Figure 5.1: Heartbeat Adaptation.

To gain understanding of how to handle these conflicts, I develop a heuristic conflict resolution 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 action, the other MCC stochastically chooses another of the candidate detailed actions that will not result in a *LRC* and applies it. It should be noted that the candidate detailed action 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 action is found, the other MCC aborts its current detailed action. If both MCCs have the same amount of data correlation, a pre-defined ordering will determine which MCC will be the first to apply its detailed action. The detailed action of the MCC that has larger data correlation is always more valuable to execute 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 with the current number of radars associated with each of them. The MCC with fewer radars receives the shared radar(s). If both MCCs have the same number of radars, a pre-defined ordering[1] is used to choose the MCC that should receive the shared radars. It is known that the more radars one MCC controls, the more time and messages it spends for local negotiation that decreases the utility for radar scanning [Krainin et al., 2007], so the resolution rule is to prefer to assign the shared radar(s) to a MCC with the lower load.

(c) Resolution Rule for IHC: To resolve an *IHC*, the rule is that the MCC with the shorter heartbeat has to adapt its communication schedule to the MCC with longer heartbeat (as Figure 5.1 shown, MCC_1 communicates with MCC_2 every two heartbeats). Since it is difficult to decide which MCC should change its heartbeat when an IHC occurs and changing the heartbeat of one MCC may

[1]The MCC that has fewer neighbors is listed in the front. In most cases, the MCCs with fewer neighbors have lower probability of being involved in conflicts compared with others.

result in other IHCs, heartbeat adaptation avoids such perplexity.

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 Chapter 5.2) 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 $a_{i,DP}$

I now describe the heuristic rule-based algorithm presented in Algorithm 2. Let $a_{i,DP}$ and $a_{j,DP}$ be the current detailed actions for MCC_i and MCC_j respectively; let $a'_{i,DP}$ and $a'_{j,DP}$ be the new detailed actions 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 Chapter 4.1 to recognize the conflicts. Each MCC_i initializes the conflict resolution process by sending messages to its neighboring MCCs (Line 1-2, Algorithm 2). I only consider the neighbors with higher index, i.e. MCC_j where $j > i$, in order to avoid resolving the conflict between two MCCs repeatedly. Each neighboring MCC then applies the heuristic rules to resolve the conflict (Line 5, Algorithm 2) if any type of conflict exists and updates the new detailed action for MCC_i and itself (Line 6-7, Algorithm 2).

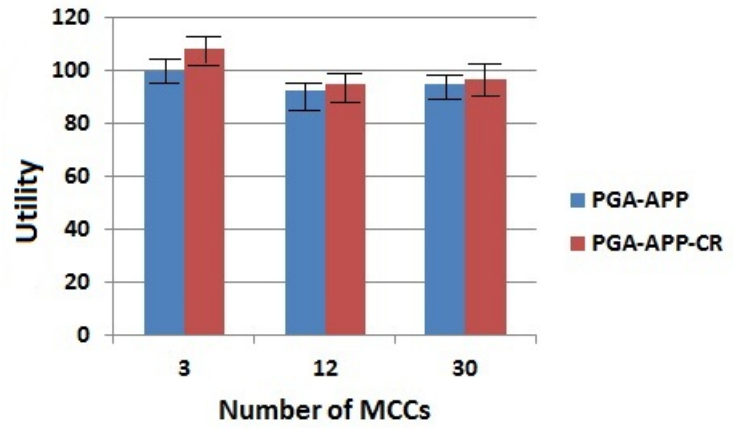
In this algorithm, each conflict is resolved by replacing the original detailed action with a new one. However, each updated detailed action does not take into account the influence it has on other neighboring agents and may introduce new conflicts with other agents. For example, in Figure 1.7, the conflict between MCC_2 and MCC_3 is resolved by changing the detailed action 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.

5.2.2 Experiments

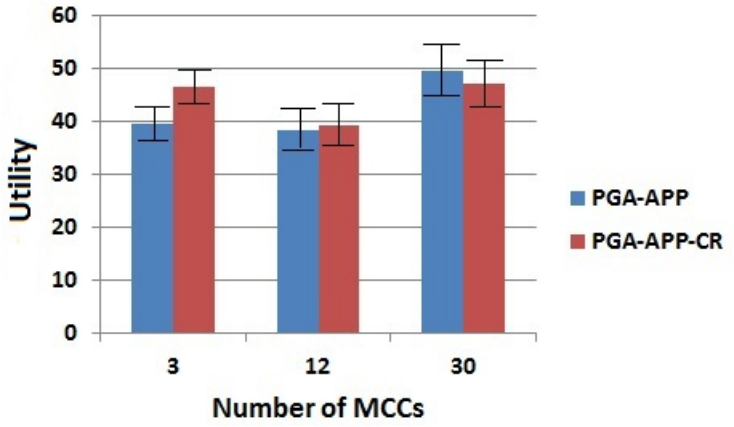
The experimental setup is as described in the previous chapter. I compare the results of two algorithms: *PGA-APP* and *PGA-APP-CR*. *PGA-APP* is the offline RL algorithm without conflict resolution as defined in the previous chapter. *PGA-APP-CR* (*PGA-APP* with *conflict resolution*) is the algorithm where I incorporate heuristic rules (defined in Chapter 5.2) to resolve local conflicts.

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, when a LRC is resolved in *PGA-APP*, the MCC with the 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 paragraphs, I empirically show that resolving conflicts explicitly improves overall system performance.*

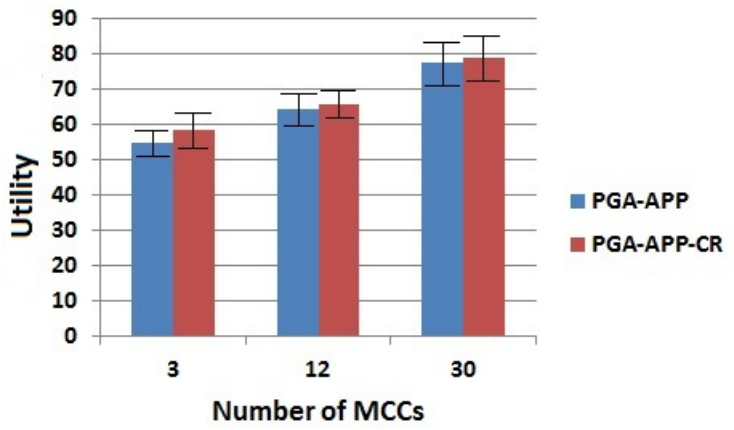
I re-ran the test cases for Figure 4.4 to compare *PGA-APP-CR* and *PGA-APP*. The results reported are the average values of 30 test episodes. Figure 5.2 shows the performance on *Utility*. Compared to *PGA-APP*, *PGA-APP-CR* achieves the larger increase with respect to *Utility* when the number of MCCs 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* is not as significant. The increase on *Utility* is only 2%, 2% and 3% 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, I can explain why the performance gain goes down with the scaling of MCCs. In fact, in the LRHS



(a) **HRLS Scenarios**



(b) **LRHS Scenarios**



(c) **MRMS Scenarios**

Figure 5.2: *Utility* of *PGA-APP* and *PGA-APP-CR* in different weather scenarios for number of MCCs to be 3, 12 and 30.

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.

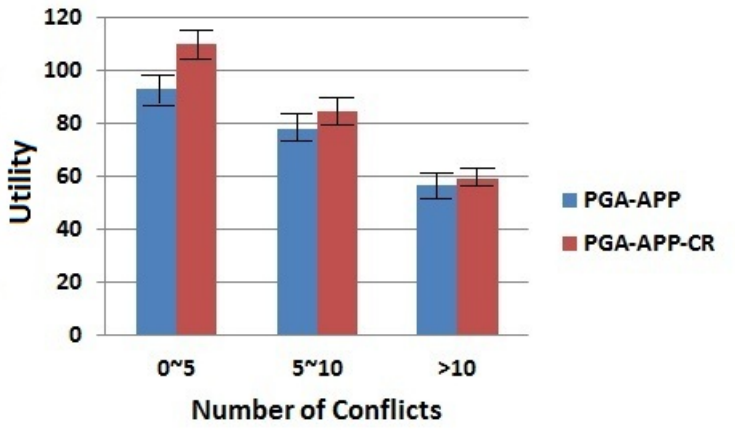


Figure 5.3: *Utility* of *PGA-APP-CR* and *PGA-APP*, number of conflicts varies.

Table 5.1: Results showing the average number of conflicts before and after conflict resolution in three categories.

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)

I ran 1000 test cases on each of the three weather scenarios where the number of conflicts were $0 \sim 5$, $5 \sim 10$ and > 10 respectively (the number of MCCs is

12, the number of tasks is 160). The results reported are average values of 30 test episodes. In Figure 5.3, I 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 the rules fades. This is mainly because the rules are defined to resolve local conflicts between two agents, and are not necessarily capable of preventing new conflicts among subsets of agents when applying the rules. Resolving local conflicts from a local perspective is not sufficient to find globally optimal solution. Figure 5.3 and Table 5.1 show the conflict resolution performance in the three different categories. 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 5.1), *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. Consider the worst case in table 5.1, the heuristic rules only resolve 44% ($1-10/18 = 1-56\% = 44\%$) of the conflicts. Let us consider the motivating example in Chapter 1 (Figure 1.7). 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 new conflict results as a consequence of the newly changed actions of MCC_1 and MCC_3 and this could not be detected in the 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.

Figure 5.4 shows the results of conflict resolution using *PGA-APP-CR* with number of MCCs increasing. As mentioned earlier, all the IHC conflicts are successfully

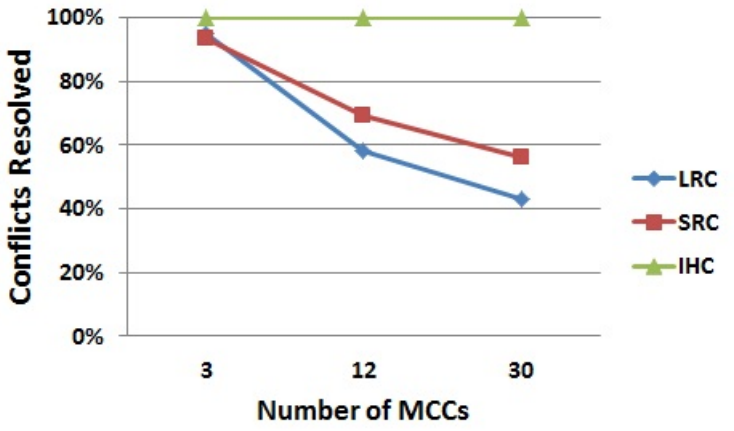


Figure 5.4: Percentage of conflicts resolved in different weather scenarios for number of MCCs to be 3, 12 and 30.

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. This is because new conflicts are resulted when resolving existing conflicts in a local view. I observe that the percentage of conflicts resolved for LRC drops below 50% when the number of MCCs reaches 30. The heuristic rule-based algorithm helps resolve some conflicts and improve the overall utility.

5.3 Coordinating Informed Unrolling and Conflict Resolution

The heuristic rule-based algorithm is capable of resolving local conflicts when there are few agents and the network is not highly constrained (In a highly constrained network, each agent has very deep dependencies with its neighboring agents.). However, the uncertainty and complexity of real-time application domains often lead to very

large state spaces for each agent as well as very deep dependencies among agents. Consider for instance, if in NetRads, a tornado tracking application [Krainin et al., 2007], I have 30 agents where each agent can have 9 neighbors. Then there are about $18 \times 19^9 \times 4^9 \approx 1.5 \times 10^{18}$ possible states for the agent to generate and reason over. Each state has three features that have 18, 19 and 4 possible values respectively, two of these features are vectors that contain the state information about neighbors. Moreover, when agents reason about deliberative actions like organization, planning, coordination etc. at the meta-level, they have to carefully choreograph the progression of what deliberations the agents should do and when. This makes decentralized online policy learning computationally challenging.

I extend the MMLC component described thus far based on the premise that when agents in a MAS have more contextual information about the states of other agents in their environment, the MAS tends to be more coordinated. Since there are costs associated with obtaining contextual information in large state spaces is significant, I claim that it is beneficial to augment agents with the capability to obtain context in an informed fashion. When conflicts between individual agents' intentions and policies are discovered as a result of this contextual knowledge, agents should also be equipped with techniques to resolve these conflicts.

5.3.1 Key Intellectual Ideas

I map these issues into two related decentralized learning research questions namely, a) how to include critical contextual information when there is a very large search space for each agent? and b) how to resolve conflicts among the learned policies of different agents? As described in Chapter 3, I model the decentralized learning problem using a decentralized Markov decision process (DEC-MDP) [Bernstein et al., 2000]. The key idea in the approach is to have the agents first learn their policies off-line within the context of a simplified environment (as defined in Chapter 4.3, the agents are assumed to experience the same weather scenario during the specific

learning episode); the agents then modify these policies on-line based on experience gained in real environments by harnessing informed unrolling and conflict resolution methods.

On-line learning is activated when conflicts resulting from multiple neighboring agents applying their local policies are observed. It is used to augment selected local policy states with additional nonlocal state information in order to learn other actions to take in this specific situation. More specifically, when an agent determines its current state, it first looks it up in its local policy, constructed off-line using multi-agent learning based on the simplistic assumption (occurring in associated off-line training scenarios) that its neighbors share its environmental context. Thus the off-line agent learning does not capture the exact environmental context of its neighboring agents in the agent state making learning much quicker since the state space is drastically reduced.

An agent operating on-line in the real domain considers the corresponding action from the off-line policy for execution. If there is no conflict with meta-level actions of neighbors as a result of this agent's proposed action or if the performance of conflict resolution is good, then the agent remains in this "normal state" S_i and executes the action prescribed by the off-line policy. Else, in the case of existence of conflicts, the agent searches the "special state" S_i' that extends its current state among the sibling states. A *special state* is a state that is added into the local policy space because of conflicts and it contains an additional state vector that captures current non-local information about neighbors. If the *special state* S_i' has not been expanded earlier, it is expanded as a sibling of the current normal state S_i. The policy space with this *special state* S_i' as root is unrolled and the policy for this subtree is used to augment the existing policy. The above steps are then repeated until the problem horizon is reached.

To balance the benefits of unrolling more states and being selective in the unrolling

direction, I use a set of heuristics (that are also part of conflict resolution negotiation strategy) and learn their priorities on-line to select actions that expand and unroll subsequent states when conflict resolution fails. The heuristics help to determine the most promising actions for overall performance improvement and unroll the states that are most likely to be encountered. I use PGA-APP to learn both the off-line policies for the initial state space as well as the on-line policies for the newly unrolled parts of the tree.

While this study focuses on meta-level questions in Netrads, I have framed the research questions to be applicable to the deliberative level as well. I believe this approach to most MAS applications where access to information about its context, improves the agents' decision making performance and makes the MAS more coordinated. The approach also benefits from the assumptions that the number of *special states* that are added via online learning are limited; and that the learned policy has a finite horizon. Conflict resolution between agent policies can be handled explicitly (as is the case for Netrads in this research) or implicitly (in domains where conflicts lead to some reduction in utility).

5.3.2 Overview of Approach

I build on the prior work where the policies are learned off-line in a simplified environment with the assumption that all agents experience the same weather scenario. I extend this work by doing on-line learning so as to learn policies in a more complicated and realistic environment where different parts of the system may encounter different types of weather scenarios. Direct application of the off-line policies to on-line scenarios could lead to undesirable performance since the off-line policies do not have the current context of the agent (now the neighboring agents could experience heterogenous weather scenarios. The tasks and data shared among agents are more complex than that in the homogeneous scenario.) that may result in conflicts. Such conflicts, if left unresolved, have detrimental effects on the overall performance of

the MAS. The on-line learning is used to augment certain local policy states with additional nonlocal state information in order to learn appropriate policies in specific situations with conflicts.

I claim that the special states *added in NetRads are limited for the following reasons: 1) It has been shown in the off-line learned policy that the states that cause conflicts are a small portion of the whole state space. The off-line policy is learned in a simplified environment that each agent experiences the same weather scenario which leads to few conflicts among neighboring agents. 2) When certain states cause conflicts, the number of conflicts could be reduced to an acceptable level (having little harmful effects to the overall performance) using decentralized negotiation algorithm that avoids the need to add in extra* special states.

This approach, called *Informed Unroll and Conflict Resolution Learning* (IU-CR-L) is described in Algorithm 3. IU-CR-L is a decentralized algorithm that learns the following issues:

(a) It learns which states cause conflicts and when "special states" need to be added based on on-line performance.

(b) It learns the priorities of the heuristics through weight updating that are used to decide which part of the search space to unroll.

(c) It learns policies for "special states" through reinforcement learning (PGA-APP).

$\psi^S(\xi)$, $\psi^I(\xi)$ and $\psi^L(\xi)$ denotes the number of SRC, IHC and LRC that exists in the neighborhood ξ respectively. *init-unroll*(*model*) unrolls the initial MDP space S^{init} for MCC_i (line 3, Algorithm 3). *compute-conflicts*(ξ) computes the number of each type of conflicts in ξ. MCC_i uses a decentralized negotiation algorithm to resolve conflicts in ξ (line 10, Algorithm 3). $PCR(\xi)$ measures the performance of conflict resolution for neighborhood ξ. $\rho(t)$ is the threshold for $PCR(\xi)$ that changes with time. *Dec-Neg-MMLC*(ξ) is the decentralized negotiation algorithm

Algorithm 3 The Learning Algorithm IU-CR-L for MCC_i

Initialize empty mdp, $initState$, $PCR(\xi)$ and $\rho(t)$;
$openList \leftarrow \{initState\}$;
$mdp =$ **init-unroll**$(model)$;
Initialize policy π as off-line optimal policy;
repeat
  Determine the weather scenario MCC_i encounters;
  Communicate and observe the current state s;
  Consider and deliberate about action a according to $\pi(s, a)$;
  $\{\psi^S(\xi), \psi^I(\xi)\ , \psi^L(\xi)\} \leftarrow$ **compute-conflicts** (ξ);
  $PCR(\xi) \leftarrow$ **Dec-Neg-MMLC** (ξ);
  if $PCR(\xi) > \rho(t)$ **then**
   Execute action a;
   Update $\pi(s)$ using PGA-APP;
  end if
  else
   Determine the *special state* s';
   if s' is not expanded earlier **then**
    Add in s' as a sibling state of s in the mdp;
   end if
   Update the current state as s';
   $a' \leftarrow$ **apply-heuristic** ();
   $a \leftarrow a'$
   $mdp =$ **Informed-Unroll** $(model)$;
   Execute action a;
   Update $\pi(s')$ using PGA-APP;
  end else
until the process is terminated.

that deliberates on the detailed actions of actions to resolve conflicts from a partially global perspective. *apply-heuristic()* is the process that uses heuristics to select an action to expand. *Informed-Unroll*$(model)$ is the smart procedure that does selective MDP expansion and negotiation so as to improve global performance.

In IU-CR-L, agents identify bad states by just "deliberating" about taking the best action at that state and predicting possible conflicts and not by actually "executing" the best action. Before an agent executes a meta-level action it contacts the agent the action will affect. If an agent that is contacted realizes through its own proposed meta-level action that there is a conflict, it will alert those agents that there is a

conflict and indicate the characteristics of the conflict (LRC, SRC or IHC). MCC_i observes its current state s, deliberates about action a to calculate the number of conflicts in its neighborhood ξ (line 7-9, Algorithm 3). Agents are synchronized and deliberate about actions at the same time. If the performance of conflict resolution is good (denoted by $PCR(\xi) > \rho(t)$), MCC_i executes a and updates the policy $\pi(s)$ and the MDP space of MCC_i remains the same (line 11-14, Algorithm 3). Otherwise, agents introduce *special states* at the same time. MCC_i searches the *special state* s' from the siblings of s. s' extends s by adding an additional state vector that contains non-local context about the neighbors. If s' has not been expanded earlier, I expand it as a sibling of s (line 17-19, Algorithm 3). Action expansion and search space unrolling in the MAS is decentralized and synchronous.

The MDP space of MCC_i with s' as root is unrolled based on interleaving of action expansion and negotiation (line 20-23, Algorithm 3). The procedure works as follows: MCC_i updates its action choice a by applying a set of heuristics in sequence that are sorted from highest to lowest priority until the performance of conflict resolution is acceptable or time has run out on this cycle. The heuristics help to direct the action of the agent in the most promising direction that improves the overall performance and more details will be provided in Chapter 5.3.6. The priority of each heuristic H_j reflects the effectiveness of H_j on conflict resolution in a specific environmental context and is learned implicitly based on multiagent reinforcement learning. I expand a and its subsequent search space if a has not been expanded earlier. At the end of the procedure, MCC_i executes a and updates the policy $\pi(s')$ (line 24-25, Algorithm 3).

5.3.3 Motivating Example

Simple scenario:

I use a couple of examples to motivate the approach. Consider a simple on-line scenario, described in Figure 5.5, where MCC_1 and all its neighbors experience the same weather scenario: HRLS. All the MCCs have the same 60 second heartbeat.

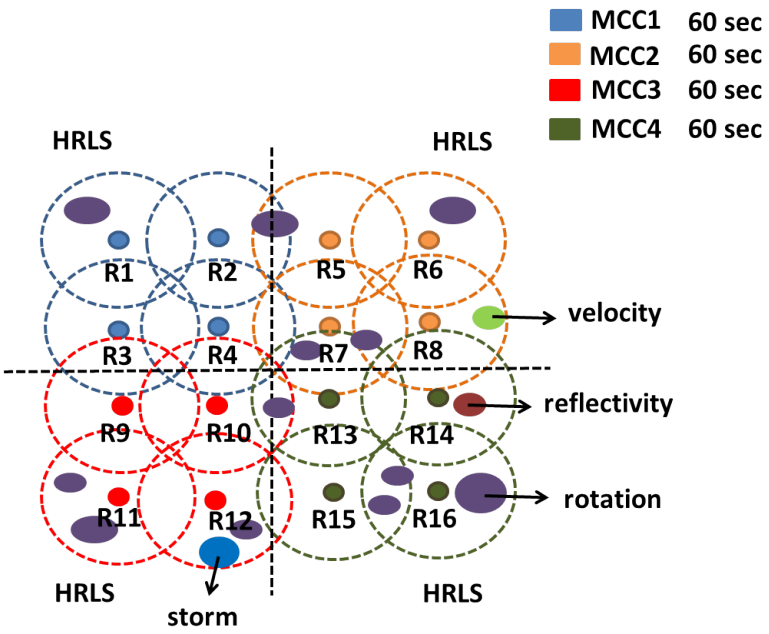


Figure 5.5: The $MCCs$ experience homogeneous weather scenarios.

MCC_1, MCC_3 and MCC_4 each have one radar involved in the data correlation while MCC_2 has three radars involved in the data correlation.

Figure 5.6 is the initial MDP space S^{init} (line 1-3, Algorithm 1) for MCC_1. At time $t = 0$ of the $MMLC$ phase, each MCC determines its current state. Figure 5.6 shows the unrolling process for MCC_1 where MCC_1 is in state S_1 and considers its action a_1 which has the highest probability distribution in the off-line policy $\pi(s_1)$. MCC_1 then computes the number of conflicts in its neighborhood (line 6-9, Algorithm

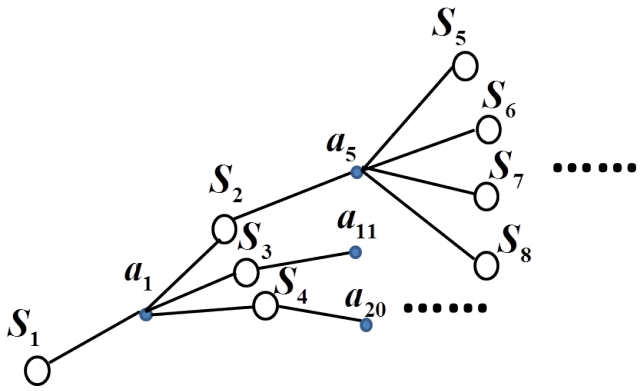


Figure 5.6: S^{init} with initial state S_1 for MCC_1 for weather scenario: HRLS.

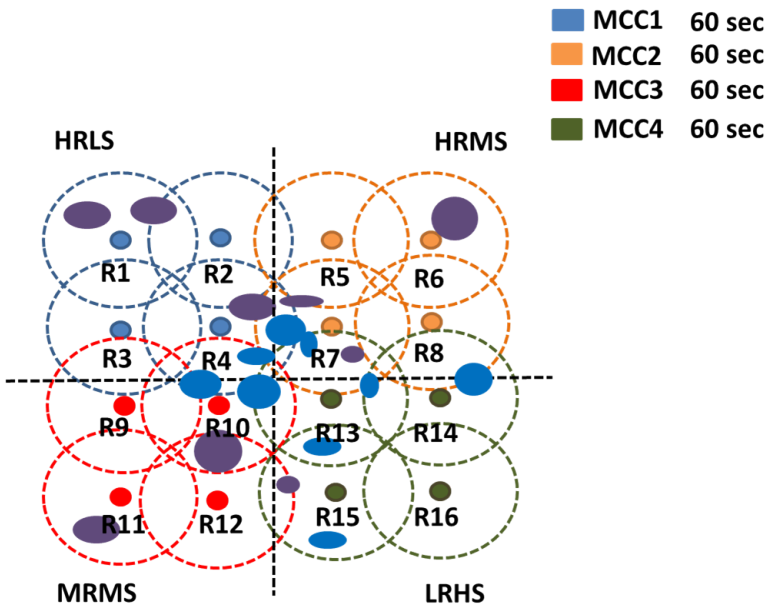


Figure 5.7: The *MCCs* experience heterogeneous weather scenarios.

1). At time $t = 0.08$ *sec*, it is observed that no conflict exists in this scenario ($\psi^S(\xi) = \psi^I(\xi) = \psi^L(\xi) = 0$; $PCR(\xi) = 1$; $\rho(0.08) = 0.784$), each MCC executes the current action and uses PGA-APP to update the policy (line 12-13, Algorithm 1). MCC_1 executes the action a_1. The state space of each MCC remains unchanged and there are no additional features added to the current state of any MCC.

Complex scenario:

I now consider a complex on-line scenario described in Figure 5.7, where MCC_1 and its neighbors experience different weather scenarios. MCC_1, MCC_2, MCC_3 and MCC_4 experiences *High Rotation Low Storm* (HRLS), *High Rotation Medium Storm* (HRMS), *Medium Rotation Medium Storm* (MRMS) and *Low Rotation High Storm* (LRHS) respectively. All the *MCCs* have the same 60 second heartbeat. MCC_1 has one radar (R_4) involved in the data correlation while MCC_2, MCC_3 and MCC_4 each have three radars (R_5, R_7 and R_8; R_9, R_{10} and R_{12}; R_{13}, R_{14} and R_{15}) involved in the data correlation.

At time $t = 0$ of the *MMLC* phase, each MCC determines its current state, con-

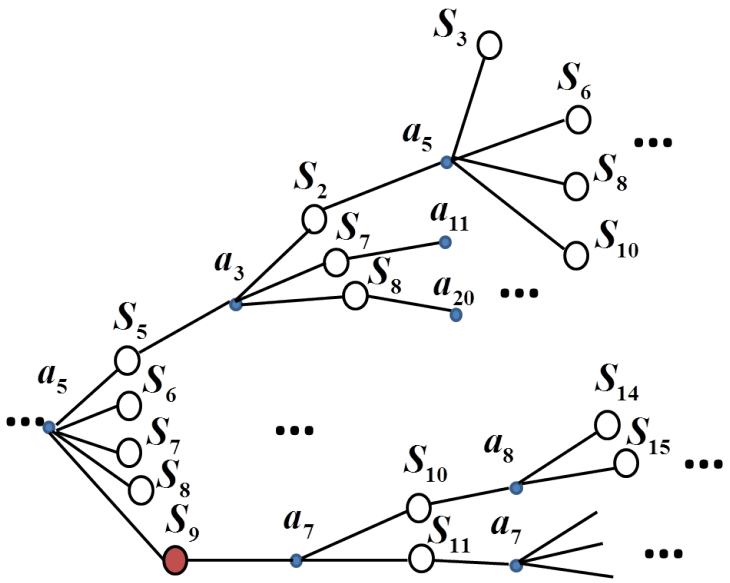


Figure 5.8: The MDP space for MCC_1 at time $t = 0.44$ *sec*.

siders its best action recommended by the offline policy and computes the number of conflicts in its neighborhood (line 6-9, Algorithm 1). Suppose MCC_1's current state is S_5 in Figure 5.8. At time $t = 0.13$ *sec*, it is observed that conflicts exist in this scenario (For the whole network: $\psi^S(\xi) = 2$, $\psi^I(\xi) = 4$ and $\psi^L(\xi) = 2$). The MCCs use the decentralized negotiation algorithm to resolve conflicts (line 10, Algorithm 1). MCC_1 deliberates about the consequences of applying various detailed actions of a_3 and chooses the detailed action that performs best on conflict resolution in its neighborhood ξ. The remaining conflicts are: $\psi^S(\xi) = 0$, $\psi^I(\xi) = 3$ and $\psi^L(\xi) = 2$. At time $t = 0.44$ *sec*, the MCCs find that the performance of conflict resolution is not good ($PCR(\xi) = \frac{4*2+2*1+1*0}{4*2+2*4+1*2} = 0.56$, $\rho(0.44) = (-0.2) * 0.44 + 0.8 = 0.712$); each MCC determines its *special state* that includes overlapping context among neighbors (line 16, Algorithm 1). This is the learning type (a) which is described in IU-CR-L.

The MCC expands the *special state* if it is not expanded earlier (line 17-18, Algorithm 1). Each MCC then stays in its *special state*, unrolls its MDP space with this *special state* as root on a selective basis (line 20-23, Algorithm 1). For MCC_1, S_9 is

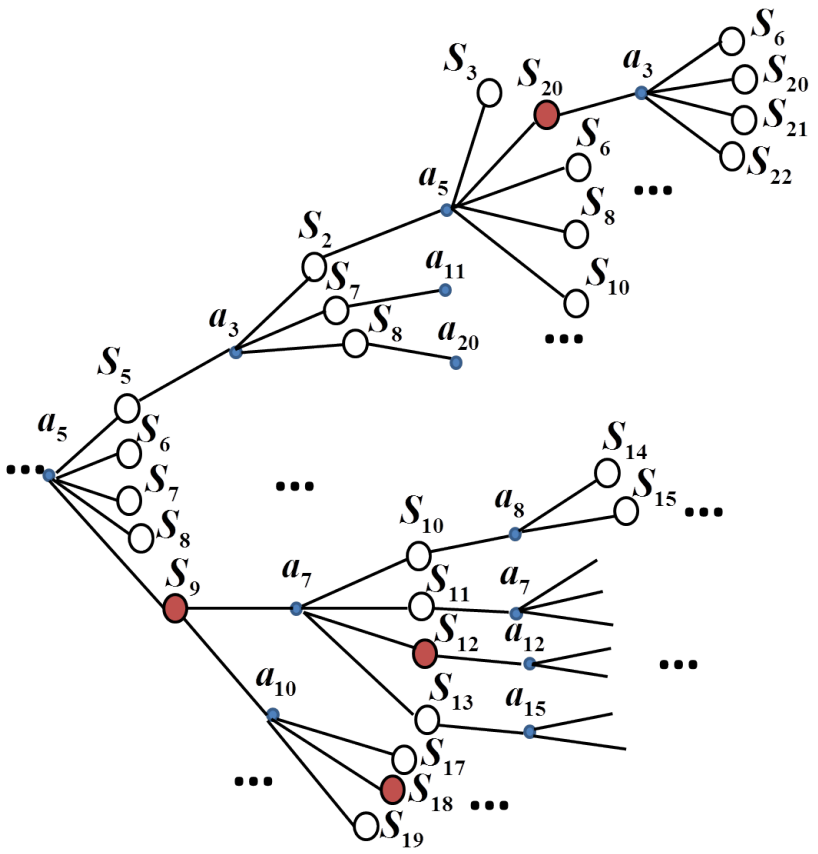


Figure 5.9: The MDP space for MCC_1 after a few learning episodes. The *special states* are marked red.

the *special state* that has not been expanded (Figure 5.8) S_9 extends S_5 by adding an additional state vector, $\langle HRMS, MRMS, LRHS \rangle$, that captures additional information about neighbors. MCC_1 applies heuristic H_1 (H_1 has the highest priority) to decide the candidate action (a_7) for expanding. Each MCC resolves conflicts with its respective new candidate action. At time $t = 1.1$ *sec*: $\psi^S(\xi) = 0$, $\psi^I(\xi) = 3$, $\psi^L(\xi) = 0$, $PCR(\xi) = \frac{4*2+2*1+1*2}{4*2+2*4+1*2} = 0.67$, $\rho(1.1) = (-0.2) * 1.1 + 0.8 = 0.58$. Since $PCR(\xi) > \rho(t)$, each MCC expands the candidate action and unrolls the MDP space with this action as root. Also, the priority value of the corresponding heuristic of each MCC is updated. This is the learning type (b) described in IU-CR-L. For MCC_1, it updates the priority value for H_1 to 0.65. The MCCs execute the newly selected actions (MCC_1 executes a_7) and use PGA-APP to update the policy $\pi(s')$

(line 24-25, Algorithm 1) which is the learning type (c) in IU-CR-L.

Figure 5.9 shows the MDP space for MCC_1 after a few learning episodes. I can see that when action a_7 fails to resolve conflicts efficiently, I use another heuristic (H_3) to choose another candidate action a_{10} to take and unrolls the subsequent search space. PGA-APP is the multiagent reinforcement learning algorithm that learns the transition function and calculates the policy. In Figure 5.9, four states (S_{10}, S_{11}, S_{12} and S_{13}) are expanded after action a_7. Initially the transition probability for these four states are evenly set values 0.25. During learning, the transition probabilities are learned, moving towards the correct ones. In the example, I observe that S_{11} and S_{12} are the two states that are visited for hundreds of times while S_{10} and S_{13} are rarely visited. The transition probabilities are updated as: $P(S_{10}|S_9, a_7) = 0.01$, $P(S_{11}|S_9, a_7) = 0.67$, $P(S_{12}|S_9, a_7) = 0.31$ and $P(S_{13}|S_9, a_7) = 0.01$.

5.3.4 Initial MDP Space

Procedure 1 $mdp =$ **init-unroll**$(model)$

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

We now describe the *init-unroll*$(model)$ in IU-CR-L that unrolls the initial MDP space S^{init} for MCC_i. Each agent obtains its S^{init} (Procedure 1) before on-line learning begins. I use the learned *Scenario Library* (Chapter 4.2.1) to determine the states and actions which are initially unrolled. S^{init} is obtained as follows: Each agent stays

in its initial state. It expands the action that has the highest probability distribution in the policy $\pi(s)$ that is stored in the *Scenario Library*(line 3, Procedure 1). Every possible state s' (I get s' by reusing the learned knowledge in the *Scenario Library*) resulting from this action is expanded (line 4, Procedure 1) and for s' similarly only the action that has the highest probability distribution in $\pi(s')$ is expanded. This process repeats until all the terminal states are reached. This greedy unrolling procedure can tell the agent what to execute (though maybe suboptimally) after the agent reaches a state during execution. *I set the depth of S^{init} to 3 which reflects the horizon of the policies for the NetRads which is three heartbeat periods. I defined this horizon manually after examining the behavior of the NetRads domain in various scenarios. It is important to establish the correct horizon, since if the horizon is too short, it triggers meta-level control too frequently which increases the cost of decision making and affects performance. On the other hand, a horizon that is too long may result in meta-level control policies that are obsolete for the latter part of the horizon, given the dynamic nature of the environment.*

5.3.5 Decentralized Negotiation

I propose a decentralized negotiation algorithm (line 10, Algorithm 3) to resolve conflicts among neighboring agents in order to better compute meta-level actions and improve the global performance. The algorithm takes into account the priorities of the three types of conflicts (SRC, IHC and LRC) when resolving them. *In Dec-Neg-MMLC, I use a mediator [Mailler and Lesser, 2006] to gradually update the detailed actions to minimize conflicts in a limited time period. Choosing agents as mediators in* IU-CR-L *helps break the conflicts into smaller sets, partially centralize and solve each set from a local view reducing communication cost.*

Prior to describing *Dec-Neg-MMLC*, I define the following key terms and functions used in the algorithm:

- r: the number of iterations of the algorithm. Each iteration includes three

Procedure 2 Dec-Neg-MMLC (ξ)

1: Initialize: $r = 1$; $C_{i,0} = \{\emptyset\}$, $C_{i,0}^{neigh} = \{\emptyset\}$; $C_{i,1}$, $C_{i,1}^{neigh}$ and P_i are computed by agents;
2: **if** $(C_{i,r} \neq C_{i,r}^{neigh})$ or $(C_{i,r-1} \neq C_{i,r-1}^{neigh})$ **then**
3: recalculate P_i;
4: **end if**
5: **for all** MCC_{neigh} **do**
6: SendMsg(P_i, MCC_{neigh});
7: CollectMsg(P_i^{neigh}, MCC_{neigh});
8: **if** $P_i \geq P_i^{neigh}$ for all MCC_{neigh} **then**
9: $mediator \leftarrow MCC_i$;
10: **branch-and-bound-search** ($mediator$);
11: $a_{i,DP}^* \leftarrow argmax_{a_{i,DP}}\{PCR(\xi)\}$;
12: **for all** MCC_{neigh} **do**
13: SendMsg($a_{neigh,DP}^*$, MCC_{neigh});
14: **end if**
15: update $C_{i,r+1}$;
16: **for all** MCC_{neigh} **do**
17: SendMsg($C_{i,r+1}$, MCC_{neigh});
18: CollectMsg($C_{i,r+1}^{neigh}$, MCC_{neigh});
19: $r++$;

stages (*Mediator Identification*, *Mediation Process* and *Configuration Update*).

- $C_{i,r} = \langle s_i, a_i, a_{i,DP}^* \rangle$: the configuration of MCC_i at iteration r is a tuple with information about MCC_i's abstract state s_i, abstract action a_i and the current detailed action $a_{i,DP}^*$ of a_i. $C_{i,r}$ is used to appropriately describe the agent's own state and action information. I set $C_{i,0} = \{\emptyset\}$ and $C_{i,0}^{neigh} = \{\emptyset\}$ to be the empty configuration before the negotiation algorithm runs. I set $C_{i,1}$ and $C_{i,1}^{neigh}$ to be the initial configuration at the beginning of the first iteration.
- $C_{i,r}^{neigh}$ has a similar definition as $C_{i,r}$, except that $C_{i,r}^{neigh}$ is the configuration of MCC_i's neighboring agent MCC_{neigh}. $neigh \in \{1, 2..., j\}$, j is the number of neighbors MCC_i has.
- $P_i = w_1 \times N_i^{SRC} + w_2 \times N_i^{IHC} + w_3 \times N_i^{LRC}$: the *mediator priority* for MCC_i that shows the priority for MCC_i to be elected as a mediator in its neighborhood. N_i^{SRC}, N_i^{IHC} and N_i^{LRC} denotes the number of SRC, IHC and LRC that MCC_i

respectively has with its neighbors. Each of these conflicts is recognized using the mechanism described in Chapter 5.1. $w_i (i = 1, 2, 3)$ is the weight for each type of conflicts. In the evaluation, I set $w_1 = 4$, $w_2 = 2$ and $w_3 = 1$ to illustrate the different degrees of importance to resolve the three types of conflicts.

- Mediator: An agent with the highest mediator priority in its neighborhood is set to be a mediator, because this agent has the most constraints within its neighborhood that leads to most powerful mediation ability. In Fig. 5.7, suppose MCC_2 has a SRC conflict with MCC_1 and a LRC conflict with the other neighbor MCC_3, the mediator priority $P_2 = 4 \times 1 + 1 \times 1 = 5$. In the NetRads domain, I assume that $P_i = 0$ if MCC_i has a high load of radars. Such an agent is not preferred to be a mediator since mediators require huge amounts of computation and communication and need to get things done quickly.
- P_i^{neigh} is defined as same as P_i, except that it is the mediator priority for MCC_i's neighbor MCC_{neigh}. $neigh \in \{1, 2..., j\}$, j is the number of neighbors MCC_i has.
- $PCR(\xi)(0 \leq PCR(\xi) \leq 1)$ measures the conflict resolution performance for neighborhood ξ (line 11, Procedure 2), it is defined as:

$$PCR(\xi) = \begin{cases} \frac{w_1 \times S'(\xi) + w_2 \times I'(\xi) + w_3 \times L'(\xi)}{w_1 \times S(\xi) + w_2 \times I(\xi) + w_3 \times L(\xi)} & \text{if } S(\xi) + I(\xi) + L(\xi) > 0 \\ 1 & \text{if } S(\xi) + I(\xi) + L(\xi) = 0 \end{cases} \tag{5.1}$$

  where $S'(\xi)$, $I'(\xi)$ and $L'(\xi)$ denotes the number of SRC, IHC and LRC that has been resolved in the neighborhood ξ respectively; $S(\xi)$, $I(\xi)$ and $L(\xi)$ denotes the number of SRC, IHC and LRC that exists before the negotiation algorithm in the neighborhood ξ respectively. w_i (i = 1, 2, 3) is the weight for each type of conflict as defined earlier. $PCR(\xi)$ takes into account both the priority and number of each type of conflicts.
- SendMsg($Content$, MCC_j): the function for MCC_i to send message to the receiver MCC_j that contains the information of $Content$.

- CollectMsg(*Content*, MCC_j): the function for MCC_i to collect message from the sender MCC_j that contains the information of *Content*.

Each iteration of the algorithm includes three stages (Stage 1: *Mediator Identification*, Stage 2: *Mediation process* and Stage 3: *Configuration update*):

1. Stage 1: Mediator Identification

   Each MCC_i computes its P_i (line 3, Procedure 2) by checking the three types of conflicts (SRC, IHC and LRC) given its configuration $C_{i,r}$ and its neighbor's configuration $C_{i,r}^{neigh}$. As mentioned earlier, P_i would be degraded to 0 if MCC_i controls a high load of radars. Note that if neither $C_{i,r}$ or $C_{i,r}^{neigh}$ changed from the previous iteration round, then P_i from the previous iteration round can be re-used without any additional computation. MCC_i then sends its mediator priority P_i to each of its neighbors (line 6, Procedure 2). It also collects messages describing P_i^{neigh} from each of its neighbors (line 7, Procedure 2).

2. Stage 2: Mediation process

   Each MCC_i receives P_i^{neigh} from each of its neighbors during Phase 1. MCC_i is set to be a mediator in its neighborhood if it has a mediator priority higher than any of its neighbors (line 9, Dec-Neg-MMLC). In the case of a tie, the MCC with lower index number is the one to be a mediator. In this way, each neighborhood has at most one mediator in each iteration round. After the mediator is set, the mediation process starts. The mediator updates a solution $a_{i,DP}^*$ using a Branch and Bound search [Zhang et al., 2005a] that efficiently resolves conflicts among the neighbors. Branch and bound is a general algorithm for finding optimal solutions of various optimization problems. It consists of a systematic enumeration of all candidate solutions by using upper and lower estimated bounds of the quality being optimized. During the search, the mediator sends each possible detailed action $a_{i,DP}$ to its neighbors along with information associated with $a_{i,DP}$ such as the current data correlation degree.

Each neighbor MCC_{neigh} sends back a message to notify the mediator whether a conflict exists and the type of the conflict if it exists. I define the following criterion to determine the solution $a^*_{i,DP}$:

$$a^*_{i,DP} \leftarrow argmax_{a_{i,DP}}\{PCR(\xi)\} \tag{5.2}$$

The criterion evaluates the performance of conflict resolution by taking into account both the number of each type of conflicts resolved as well as the importance of each type of conflicts. The mediation result may include the changes of the detailed actions of abstract actions of the neighbors. The newly updated actions are sent to the neighbors by the mediator (line 13, Dec-Neg-MMLC).

It is not guaranteed that all the conflicts (between MCC_i and its neighbors) can be successfully resolved as a result of the mediation process. The space of the detailed actions of the abstract actions of MCC_i is limited and the resolution of one conflict influences the future conflict resolutions (The change of $a^*_{i,DP}$ may change the conflict types of other conflicts in the neighborhood, or introduce new conflicts into the neighborhood). In some cases, there is no mediator in a certain neighborhood when each agent in this neighborhood has a lower mediator priority than some neighbor(s) outside this neighborhood. After some iteration rounds, the mediator priorities of the agents outside this neighborhood decrease and become lower than those of the agents inside the neighborhood. The conflicts in this neighborhood would then be resolved by electing one mediator inside it.

3. Stage 3: Configuration update

   Each MCC_i updates its configuration $C_{i,r+1}$ (line 15, Procedure 2) and sends out messages describing $C_{i,r+1}$ to all of its neighbors (line 17, Procedure 2). This step is necessary since after Phase 2, most MCCs have changed their configurations. Meanwhile, MCC_i collects the messages from its neighbors and

populates $C^{neigh}_{i,r+1}$.

The *Dec-Neg-MMLC* algorithm transmits a linear number of messages for information updates. At each iteration r, MCC_i sends messages to its neighbors describing $C_{i,r}$, P_i and updated meta-level information only once. The largest number of messages for information updating one MCC sends/receives at each iteration depends on the largest neighborhood size in the problem. The largest portion of messages comes from Stage 2. During the Branch and Bound search, the mediator exchanges messages with its neighbors for each possible detailed action. The communication cost increases substantially when the mediator has a huge number of candidate detailed actions.

5.3.6 Informed Unrolling

Procedure 3 mdp = **Informed-Unroll**($model$)

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

Informed-Unroll (line 23, IU-CR-L) is the process that selectively expands the MDP space based on the performance of iterative conflict resolution when MCCs reach *special states* (line 23, Algorithm 1). *partial-unroll*($s', a, model$) expands action a and the subsequent search space in a similar way as *init-unroll*($model$). When *Informed-Unroll* starts, each agent checks the *termination condition* (line 1, Procedure 3) to decide whether the *MMLC* phase needs to be terminated or not. I set the following rules as the *termination condition* for the *MMLC* phase:

1. When no conflict exists among the meta-level actions of each MCC, there is no need to run the decentralized negotiation algorithm in the *MMLC* phase and the time saved in the *MMLC* phase is allocated to Phase 3 and 4 for better performance.
2. When conflicts exist among the meta-level actions of each MCC, allocating as much time as possible (should not exceed the time limit for the *MMLC* phase) to the decentralized negotiation algorithm for conflict resolution. If the time limit for the *MMLC* phase is reached or the conflict resolution performance is good ($PCR(\xi) > \rho(t)$), the negotiation will terminate.

I use a linear function to calculate the actual threshold $\rho(t)$ (line 3, Procedure 3) for $PCR(\xi)$ (t is the amount of time spent (sec) in the *MMLC* phase, $0 \leq t \leq 3$.) as:

$$\rho(t) = -a \cdot t + b \tag{5.3}$$

where a and b are constants. At the start of the *MMLC* phase, $\rho(t)$ is high so that the agents have higher probability to select an optimal action that minimizes conflicts. During the latter part of the *MMLC* phase, it is better to decrease $\rho(t)$ so that the agents are risk-averse to poor conflict resolution performance without enough time left to fix it. Since *MMLC* phase is time constrained, I set the upper bound of t to be 3 ($30 \times 10\% = 3$) with respect to the 30 seconds heartbeat.

I use heuristics to expand appropriate actions (line 10, Procedure 3) in order to direct the agent in the most promising direction that improves the overall performance to the ultimate extend in limited time. The intuition behind the heuristics is that each heuristic works well on conflict resolution under a certain condition. I learn the priority of each heuristic on-line and use the heuristics to decide which action to take when a special state *is encountered.* I define heuristics as:

- H_1: The agent unrolls its MDP space by exploring a new action that has a different *type* of radar movement from the current action choice. This heuristic helps in situations that there are lots of data correlation existing among

overlapping areas. For example, H_1 makes MCC_1 to change the action 'Heavy Move (MCC_1 *to* MCC_2)' to 'Light Move (MCC_1 *to* MCC_2)'.

- H_2: The agent unrolls its MDP space by exploring a new action that has a different *direction* of radar movement from the current action choice. This heuristic helps in such situations that the agent moves radars to its heavily loaded neighbor(s) while keeping other neighbor(s) free. A heavily loaded MCC [Krainin et al., 2007] is one that reaches the upper bound of controlling radars. For example, H_2 makes MCC_1 to change the action 'Heavy Move (MCC_1 *to* MCC_2)' to 'Heavy Move (MCC_1 *to* MCC_3)'.
- H_3: The agent unrolls its MDP space by exploring a new action that has a different *heartbeat* from the current heartbeat choice. This heuristic helps in situations that the agent has a different heartbeat with most of its neighbors.
- H_4, H_5 and H_6 are the heuristics that enforce two of the above three elementary heuristics. For example, H_4 is the heuristic that the agent unrolls its MDP space by exploring a new action that has a different type as well as a different direction of radar movement from the current action choice.
- H_7 enforces H_1, H_2 and H_3 simultaneously.
- H_8 is used to denote that an agent does not change its current action.

Initially, we have no prior knowledge about which heuristic works best for MCC_i under a *special state*. For MCC_i, I define $\chi_{i,j}$ $(j = 1, 2, ..., 8)$ as the priority that measures the effectiveness for applying heuristic H_j for MCC_i.

update-heuristic-priority() updates the priority for each heuristic based on the actual performance on conflict resolution (line 7, Procedure 3):

$$\chi_{i,j} \leftarrow \frac{\chi_{i,j} \times N_{i,j}^{sum} + PCR(\xi)}{N_{i,j}^{sum} + 1} \tag{5.4}$$

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

MCC_i sorts all the heuristics in a descending order based on $\chi_{i,j}$ and put them on a list. Each time when conflict resolution fails ($PCR(\xi) \leq \rho(t)$), MCC_i re-

selects action a' (line 10, Procedure 3) by applying the heuristic that has the highest priority in the remaining list. When a heuristic has been applied, it will be removed from the list. Each heuristic has the chance to be taken in this way. When conflict resolution is successful ($PCR(\xi) > \rho(t)$), the priority of each heuristic is updated using Equation 5.4 and all the heuristics are put back to the list and re-sorted. The heuristic with lower priority previously could increase its priority if it continuously has good performance on conflict resolution.

5.3.7 Experiments

I evaluate the algorithm *IU-CR-L* on scenarios with 12 agents controlling a total of 72 radars where each scenario contains heterogenous weather scenarios in different parts of the system based on the distribution of the tasks. A task represents a weather event and I am only concerned about *rotation* and *storm* tasks in the evaluation. The number of tasks varies from 80 to 200 for each scenario. There are nine types of possible weather scenarios occurring in the system, and they are differentiated by the number of these two tasks. Each MCC has two choices of heartbeat: 30 seconds long or 60 seconds long.

I compare the results of four approaches: *IU-CR-L*, *IU-CR-H*, *IU-CR* and *No-IU*. *IU-CR-L* is the online learning approach that iteratively expands the MDP search space, uses a decentralized negotiation algorithm to resolve conflicts, updates the priorities of heuristics (H_1 to H_8) and uses PGA-APP to update the policy. *IU-CR-H* is the online learning approach that uses heuristics to unroll the MDP search space. The priorities of the heuristics are also learned online. *IU-CR-H* does not use PGA-APP to update the policy. *IU-CR* is different from *IU-CR-L* in that it expands all the action choices instead of using heuristics to selectively unroll the promising action choice when conflicts occur. *No-IU* is the online learning approach that takes advantage of previously learned off-line policy from the *Scenario Library*. It uses PGA-APP to update the policy without conflict resolution. In the experiments,

$\rho(t) = -0.2 \cdot t + 0.8$.

In the following paragraphs, I empirically show that IU-CR-L learns useful policies for agents with a small amount of training episodes. Also, IU-CR-L achieves significantly better performance on utility and conflict resolution by unrolling a small fraction (only 10% *in the best cases) of the whole search space.*

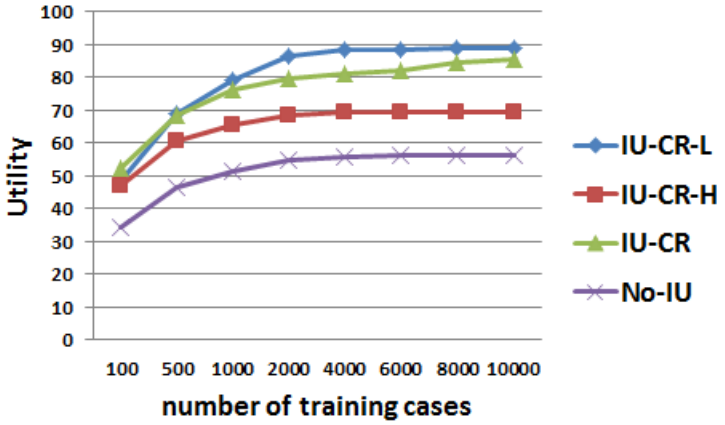


Figure 5.10: *Utility* of the four algorithms for 12 MCCs, for various number of training episodes.

I used a total of 10000 training episodes to learn the policies for the four algorithms. I gradually increased the number of training episodes (as Figure 5.10 shows) and used 30 test cases to evaluate the performance of policies that converge for different amounts of training episodes. In Figure 5.10, I observe that *IU-CR*, *IU-CR-H* and *IU-CR-L* perform significantly better than *No-IU* on *Utility* for all the training episodes. Conflict resolution helps to improve the overall performance on *Utility*. I also observe that *IU-CR* performs better than *IU-CR-L* on *Utility* when 100 training episodes are run, this is because the heuristics with inaccurate priorities may lead to biased expansion of actions. As the number of training episodes increases, the state space of each agent using *IU-CR-L* is more accurate about the neighbors' environmental states; and the conflict resolution actions have been frequently explored and executed. These two factors lead to improved performance with *IU-CR-L* with increased training.

PGA-APP is useful and it helps to learn the appropriate policy that improves the overall performance significantly.

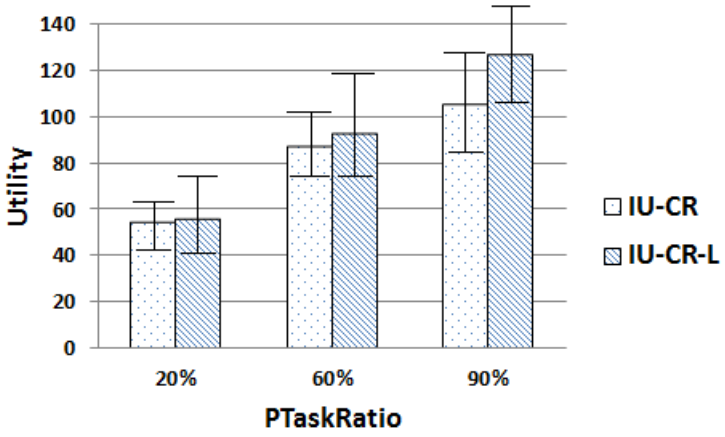
I varied *PTaskRatio* (setting it to 20%, 60% and 90%) and compared the performance of policies. When radars belonging to different MCCs share data (especially data about shared pinpointing tasks), the communication among these MCCs would increase and thus there is more interdependency. The pinpointing tasks need significant coordination among MCCs. In Figure 5.11(a), I note that *Utility* increases with the increase of the percentage of pinpointing tasks for both approaches. *IU-CR-L* improves 3%, 7% and 21% for the 20%, 60% and 90% *PTaskRatio* cases respectively on *Utility* compared with *IU-CR*.

Figure 5.11(b) and Figure 5.11(c) show that *IU-CR*, *IU-CR-H* and *IU-CR-L* perform significantly better than *No-IU* on *Time* for the 20%, 60% and 90% *PTaskRatio* cases respectively. For *IU-CR*, p values are 0.0043, 0.018 and 0.047 respectively; for *IU-CR-H*, p values are 0.0038, 0.039 and 0.044 respectively; for *IU-CR-L*, p values are 0.029, 0.001 and 0.0072 respectively. Adaptive meta-level control allows for effective use of the heartbeat. By coordinating meta-level control parameters, the *Negotiation* phase converges more quickly, so that more time can be spent on the *Data Processing* and *Local Optimization* phases. In *IU-CR*, *IU-CR-H* and *IU-CR-L*, the negotiation algorithm helps agents resolve conflicts and select the most appropriate action choices from a non-local perspective to reduce the time in the *Negotiation* phase.

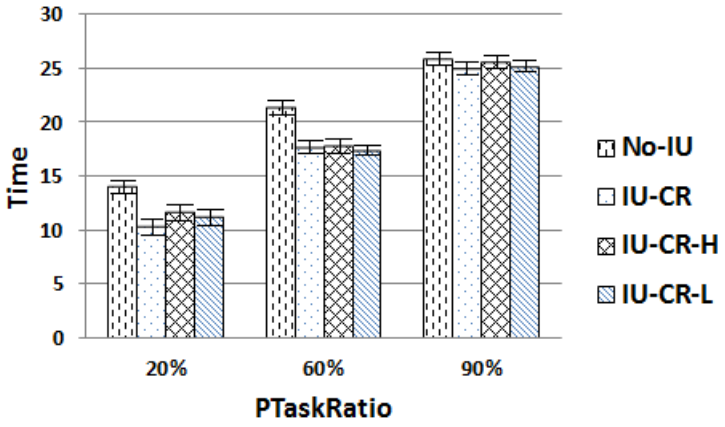
Table 5.2: Comparison results between *IU-CR* and *IU-CR-L* after 1000 training episodes.

Approach	# of states expanded	# of *special states* expanded	# of newly learned policy	Unrolling time (sec.)
IU-CR	21389271	214	214	1.56
IU-CR-H	3702904	152	152	0.084
IU-CR-L	3587659	137	137	0.084

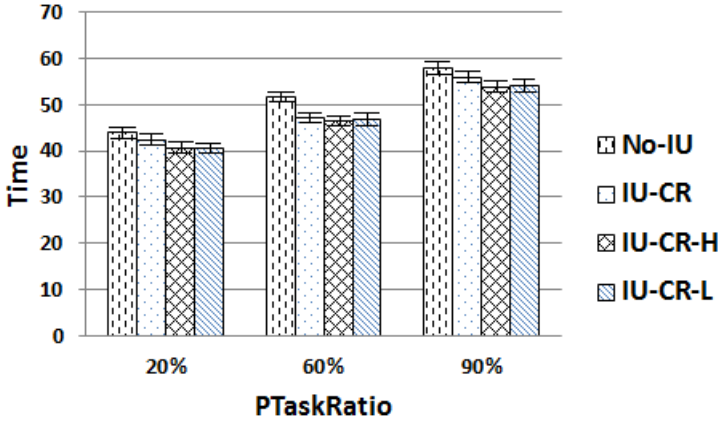
Figure 5.12 shows that *IU-CR* and *IU-CR-L* achieve similar conflict resolution



(a) *Utility*



(b) 30 seconds heartbeat



(c) 60 seconds heartbeat

Figure 5.11: Performance for 12 MCCs, for *PTaskRatio* to be 20%, 60% and 90%.

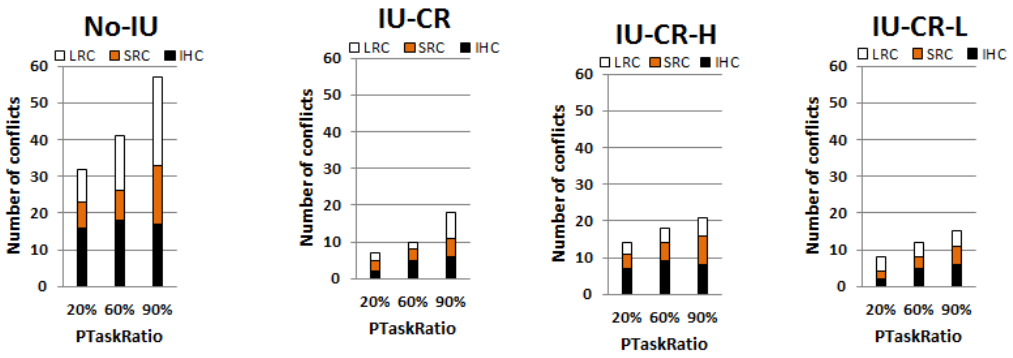


Figure 5.12: Number of conflicts (LRC, SRC and IHC) unresolved by the four algorithms for 12 MCCs, for *PTaskRatio* to be 20%, 60% and 90%.

performance. *IU-CR* reduces the number of conflicts unresolved by 78%, 76% and 68% for the 20%, 60% and 90% *PTaskRatio* cases respectively compared with *No-IU*; while *IU-CR-L* reduces the number of conflicts unresolved by 75%, 71% and 74% for the three cases respectively.

Table 5.2 shows that for *IU-CR*, *IU-CR-H* and *IU-CR-L*, the *special states* added to the search space are limited. *IU-CR-H* and *IU-CR-L* unroll significantly fewer states than *IU-CR* (reducing 95.5% and 96.8% respectively). Among these expanded states only a small fraction ($< 0.01\%$) are visited during learning. Each learned sub-policy associated with an encountered *special state* is iterated 437 times on average. In *IU-CR-L*, the heuristics help to balance the benefits of unrolling more states and of being selective in the unrolling direction. Although *IU-CR-L* learns fewer sub-polices compared with *IU-CR* and *IU-CR-H*, its policies perform better on *Utility* (see Figure 5.10). *IU-CR-L* suggests a new action for a *special state* at the probability of 31.8%; while *IU-CR-H* does so at the probability of 13.5%. In *IU-CR-L*, PGA-APP contributes to the learning of the expanded actions and thus leads to the appropriate sub-policies. *IU-CR* spends significantly more time (1.56 *sec* compared with 0.084 *sec*) on unrolling compared with *IU-CR-H* and *IU-CR-L*. This is because when *IU-CR* reaches a *special state*, it expands all the possible actions and unrolls the resulting

search space of the actions that consumes much time.

Table 5.3: Comparison results of *IU-CR-L* with the increase of training episodes.

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

Table 5.3 shows that the percentage of new states expanded using *IU-CR-L* is decreasing substantially with the increase of training episodes. During the earlier learning stage (from 100 to 500 training episodes), many new *special states* are encountered that results in a huge unrolling space. After that, the probability to expand new *special states* decreases. *IU-CR-L* expands 281% and 19.1% more states when number of training episodes increases to 500 and 1000 respectively.

During the earlier learning stage, the dominant heuristics are more likely to be chosen, so the average number of actions expanded after *special states* is low. When more training episodes are encountered, the number of heuristics that are applied increases because of the uncertainty of conflict resolution. So the average number of actions expanded also increases (5.6 compared with 2.3 in Table 5.3). After enough training episodes are encountered, the priority of each heuristic becomes accurate and the *special states* expanded later in the on-line learning have higher probability to apply only the few dominant heuristics. For this reason, the average number of actions expanded after *special states* decreases (5.1, 4.6 and 3.4 respectively compared with 5.6 in Table 5.3). *IU-CR-L* learns the effectiveness of applying each heuristic.

5.3.8 Discussion

It is guaranteed that for the same set of conflicts, the negotiation algorithm changes the solution towards less conflicts solution gradually. Suppose MCC_i is elected as a mediator at iteration r. After the mediation process of MCC_i, the sum of mediator

priorities in MCC_i's neighborhood decreases (it is equal to the sum in previous iteration $r-1$ at the worst case). The purpose of the mediator is to alleviate the degree of conflicts in its neighborhood, thus lowering the sum of mediator priorities in the neighborhood. Consequently the sum of mediator priorities of all MCCs is decreased (or unchanged in the worst case) at the end of each iteration. So the solution is moving towards less conflicts one gradually at each iteration. The question now is to determine how close is the final solution to the optimal one? It depends mainly on the following three factors:

1. Number of saturated neighborhoods: *Saturated neighborhood* is the neighborhood in which the degree of conflicts is minimized and can not be decreased further by the mediation process. When a neighborhood becomes a *saturated neighborhood*, the sum of mediator priorities in it keeps unchanged at the end of each iteration. The *saturated neighborhood* could become unsaturated by resolving the conflicts from a non-local perspective. The negotiation algorithm only resolves conflicts locally (the scope of the neighborhood) and can not handle this issue. The speed of converging to the optimal solution depends on the total number of *saturated neighborhoods* in the problem. The larger the number of *saturated neighborhoods* is, the slower the speed is.
2. Number of negotiation rounds: The actual number of negotiation rounds in the algorithm depends on the factors such as the total number of MCCs, the average size of neighborhoods in the problem, the degree of conflicts in each neighborhood and the actual time spent on the mediation process. The negotiation round in a larger neighborhood takes more time compared with a smaller one in terms of more constraints to consider and more messages to transmit. The more number of negotiation rounds is in a specific scenario, the more closer is the final solution to the optimal one.
3. Degree of conflict resolution difficulty: In some scenarios, the conflicts in each

neighborhood have few or no dependencies with those in other neighborhoods. The conflicts could be efficiently resolved (or partially resolved) by the mediation process at each negotiation round. The solution moves closer to the optimal one at each negotiation round smoothly. The negotiation algorithm is preferred in such scenarios. However, in other scenarios that the conflicts in each neighborhood have high dependencies with those in other neighborhoods. Resolving such conflicts locally at each negotiation round improves the performance very slowly. In these scenarios, the negotiation algorithm may perform badly that calculates the solution being far away from the optimal one.

5.4 Global Optimization

In the previous section, I described the decentralized negotiation algorithm that is used to resolve conflicts from a partially global perspective. The algorithm is not guaranteed to resolve all the conflicts in the problem (as shown in the experimental results). In this chapter, I will delve into details on how decentralized learning and DCOP algorithm are synthesized to resolve conflicts and reach globally optimal solution. An empirical study is presented for different scenarios and the strengths and weaknesses of a global optimization approach is presented.

5.4.1 Applying Max-sum to Resolve Conflicts Globally

I formulate the MMLC conflict resolution coordination problem discussed in the previous Chapter as a DCOP, which is defined by a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, where

- $\mathcal{A} = \{A_1, ..., A_k\}$ is a set of agents; in the NetRads context, each MCC_i is assigned an agent.
- $\mathcal{X} = \{X_1, ..., X_m\}$ is a set of m discrete variables; in the NetRads context, each MCC_i has a variable X_i, which represents MCC_i's *abstract actions* defined in Chapter 3.
- $\mathcal{D} = \{d_1, ..., d_p\}$ is a set of domains for the variable set $\mathcal{X}$. In the NetRads context, the domains are the detailed actions of *abstract actions* for each MCC.

- $\mathcal{F} = \{F_1, ..., F_n\}$ is a set of n functions; each function $F_i(\mathcal{X}_i)$ is dependent on a subset of variables $x_i \subseteq \mathcal{X}$ defining the relationship among the variables in x_i. In the NetRads context, $F_i(\mathcal{X}_i)$ is defined as:

$$F_i(\mathcal{X}_i) = \sum_{MCC_m \in C_i} CP_m(a_m^n) \tag{5.5}$$

where $\sum_{MCC_m \in C_i} CP_m(a_m^n)$ is the sum of conflict penalties of each possible assignment of the variables involved in the neighborhood of the MCC_i. C_i is defined as the MCC configuration in the neighborhood of MCC_i. a_m^n is the nth possible detailed action of the abstract action a_m. The conflict penalty of applying the detailed action a_m^n is defined as:

$$CP_m(a_m^n) = -|w_1 \times N_m^{SRC} + w_2 \times N_m^{IHC} + w_3 \times N_m^{LRC}| \tag{5.6}$$

where N_m^{SRC}, N_m^{IHC} and N_m^{LRC} denotes the number of SRC, IHC and LRC MCC_m has with its neighbors respectively. As mentioned earlier, each of these conflicts is recognized using the mechanism, described in Chapter 5.1. $w_i (i = 1, 2, 3)$ is the weight for each type of conflicts, which has the same value as defined in Chapter 5.3.4. The conflict penalty $CP_m(a_m^n)$ measures the severity of conflicts MCC_m will bring into the problem when the detailed action a_m^n is applied. The lower $CP_m(a_m^n)$ is, the more conflicts MCC_m will bring into the problem. $CP_m(a_m^n) = 0$ means MCC_m has no conflict with its neighboring MCCs if the detailed action a_m^n is applied.

The optimization problem here is defined as finding the detailed actions for radar reorganization and heartbeat adaptation that maximizes the sum of $F_i(\mathcal{X}_i)$ in the system. In other words, the goal is to find the detailed actions of abstract actions for MCCs that maximizes the performance of conflict resolution in the whole system.

In this work, I use the Max-sum algorithm to compute coordinated joint policies for both learning and execution stages. During the learning stage, I extend the

Algorithm 4 The Learning Algorithm IU-CR-L' for MCC_i

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

IU-CR-L algorithm (presented in Chapter 5.3.1) by replacing the decentralized negotiation algorithm with Max-sum when agents deliberate about their actions (line 10, Algorithm 4). If the performance of conflict resolution is good, MCC_i executes its meta-level action a and updates the policy $\pi(s)$ and the MDP space of MCC_i remains the same (line 11-14, Algorithm 4). Otherwise, *special state* s' (line 16-20, Algorithm 4) is introduced and there is no further exploration (the state space of MCC_i is not expanded). Next time MCC_i observes the *special state* s', it will choose a different meta-level action a' and uses Max-sum to find the best detailed action (line 21-22, Algorithm 4).

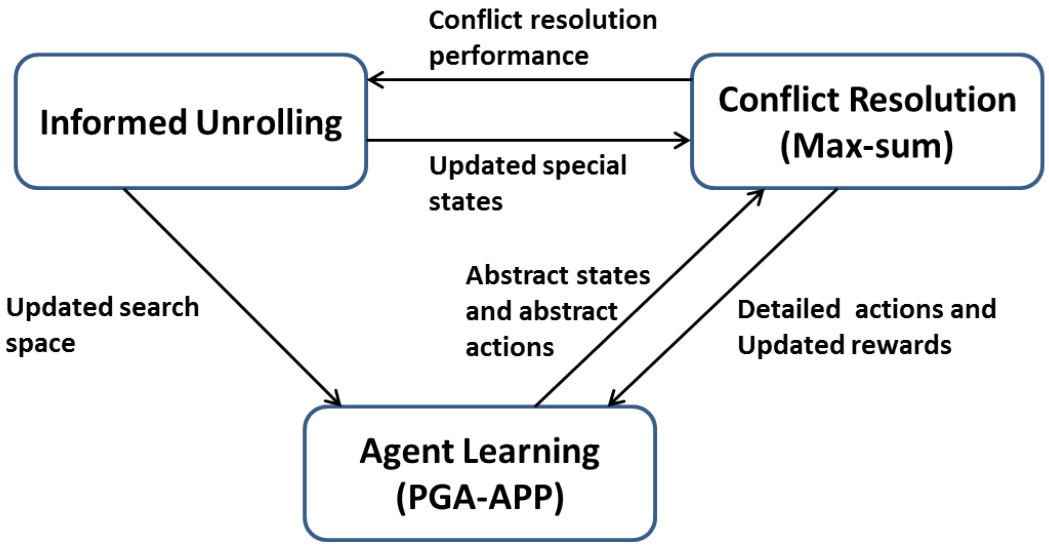


Figure 5.13: Framework flow diagram for algorithm IU-CR-L'.

As mentioned, when the factor graph of Max-sum is not cycle free, the convergence is not guaranteed and Max-sum continues running by increasing rounds of passing messages. I set the following rules as the *termination condition* for Max-sum in case it exceeds the time limit of the *MMLC* phase ($\leq 10\%$ of the heartbeat):

1. When no conflict exists among the meta-level actions of each MCC, there is no need to run the Max-sum algorithm and the time saved in the *MMLC* phase is allocated to Phase 3 and 4 for better performance.
2. When conflicts exist among the meta-level actions of each MCC, allocating as much time as possible (should not exceed the time limit for the *MMLC* phase) to run Max-sum. If the time limit for the *MMLC* phase is reached and Max-sum is still running (not converged), Max-sum will be terminated and approximate solutions are utilized.

Figure 5.13 is the framework flow diagram illustrating the relationships among the three key components in IU-CR-L'. The Max-sum algorithm serves two main functionalities: a) It determines the best detailed actions along with the reward r_{CR} based on the current abstract states and actions it receives from PGA-APP b) It supervises the informed unrolling process on whether to stay in the current MDP

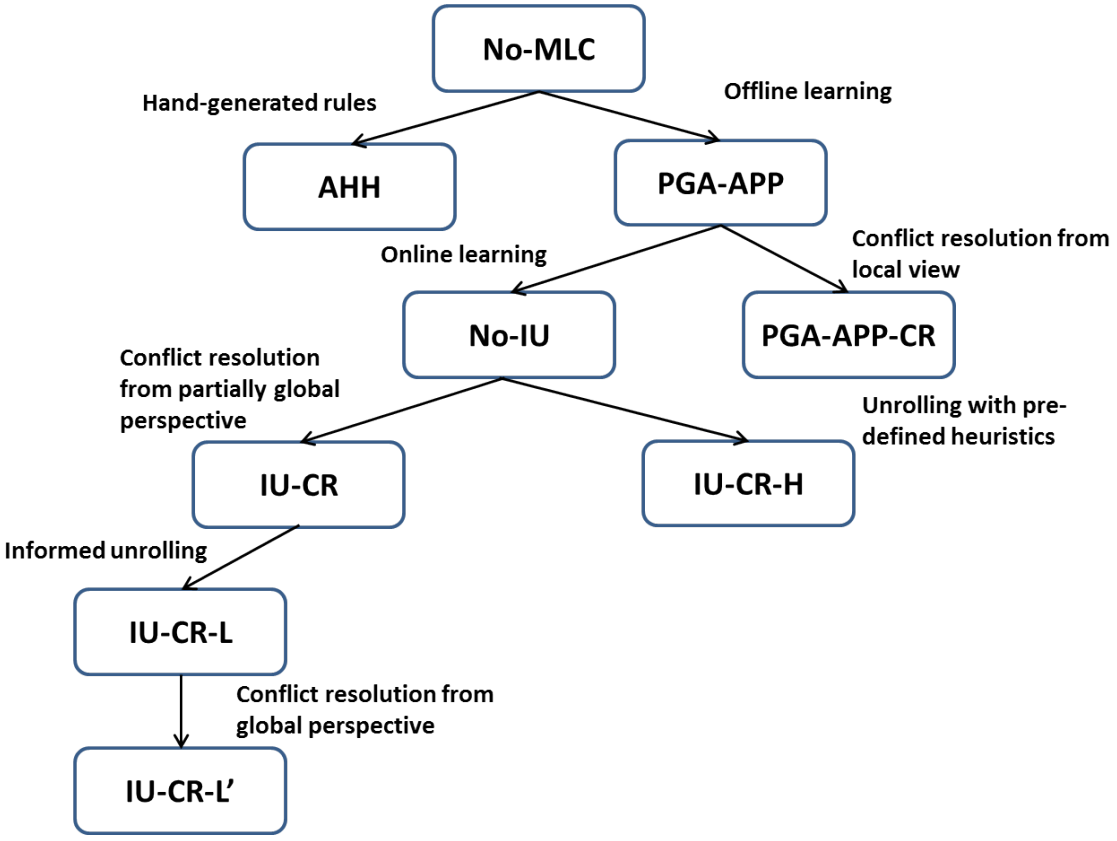


Figure 5.14: The relationships between all the algorithms.

space or to expand further MDP space by adding in new special states. Max-sum chooses reward $r_{CR} = \max_{\forall n} CP_m(a_m^n)$ for local abstract action a_m of agent MCC_m and sends it back to PGA-APP. The reward r_{CR} reflects the performance of conflict resolution by using abstract action a_m and needs to be included in the observed reward r (line 5, Algorithm 1) in PGA-APP. Figure 5.14 illustrates the relationships between all the algorithms developed in this dissertation.

5.4.2 Experiments

I evaluate the Max-sum-based approach using the same simulation environment of the NetRads system as described in Chapter 4.4. I use the same training/test cases discussed in Chapter 5.3.6 and compare the results of five approaches: *IU-CR-L'*, *IU-CR-L*, *IU-CR-H*, *IU-CR* and *No-IU*. *IU-CR-L'* is the learning approach that iteratively expands the MDP search space, uses Max-sum algorithm to resolve conflicts

from a global perspective, updates the priorities of heuristics and uses PGA-APP to update the policy. *IU-CR-L'* differs with *IU-CR-L* in that *IU-CR-L'* is implemented with the Max-sum algorithm to resolve conflicts from a global perspective while *IU-CR-L* is implemented with the decentralized negotiation algorithm that resolves local conflicts.

For experiments involving the decentralized negotiation algorithm as well as the Max-sum algorithm, each MCC, when the local computation is completed, waits for other MCCs to finish the computation and then they exchange messages. Therefore, the time complexity in the decentralized setting results from the sum of the longest time taken in the local computation for each round. It is assumed there is no communication delay in measuring the completion time. Both the number of messages and the size of messages were measured to compute communication costs and this includes counting the control messages to construct the network as well including the time for establishing connectivity between nodes and information sharing on possible values that each variable can take. The total amount of communication is measured in bytes.

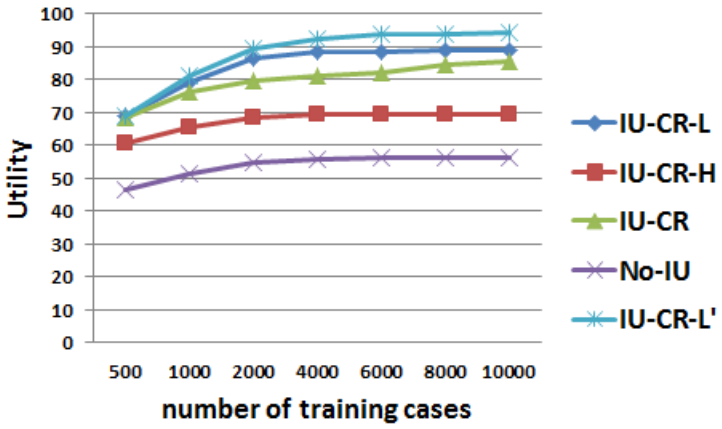
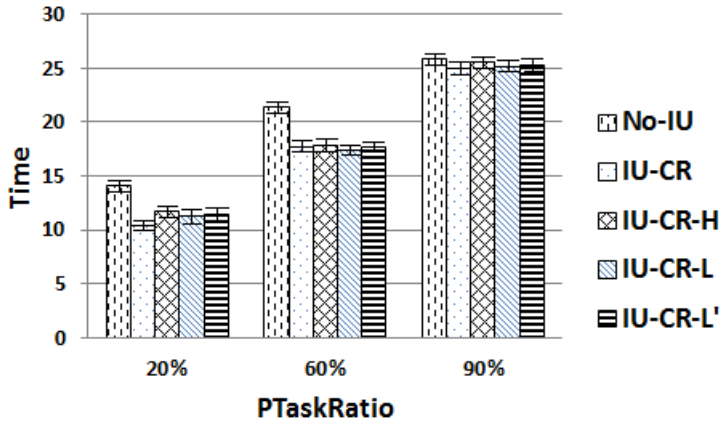
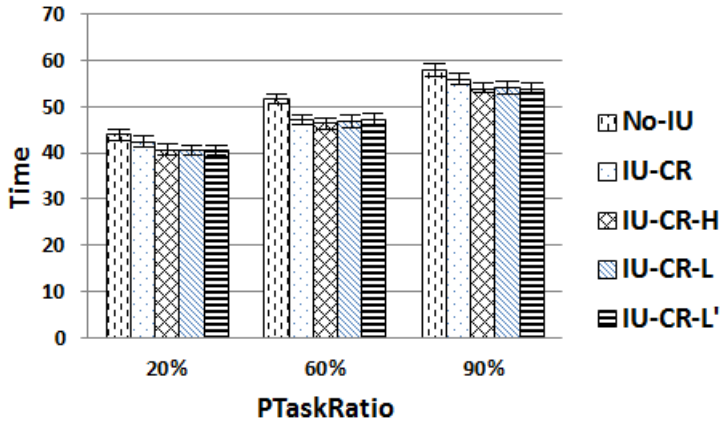


Figure 5.15: *Utility* of the five algorithms for 12 MCCs, for various number of training episodes.

In Figure 5.15, I observe that *IU-CR-L'* provides the upper bound for *Utility* among



(a) 30 seconds heartbeat



(b) 60 seconds heartbeat

Figure 5.16: *Time* of the five algorithms for 12 MCCs, with *PTaskRatio* of 20%, 60% and 90% respectively.

these five algorithms. *IU-CR-L'* improves 0.3%, 2.1%, 3.5%, 4.2%, 5.8%, 5.9% and 5.7% respectively on *Utility* compared with *IU-CR-L*. The Max-sum algorithm helps resolve more conflicts and thus calculate the better policy. Figure 5.16(a) and Figure 5.16(b) show that *IU-CR-L'* performs significantly better than *No-IU* on *Time* for the three cases respectively. p values are 0.019, 0.0084 and 0.027 respectively. The Max-sum algorithm helps agents resolve conflicts and select the most appropriate action choices from a global perspective to reduce the time in the *Negotiation* phase.

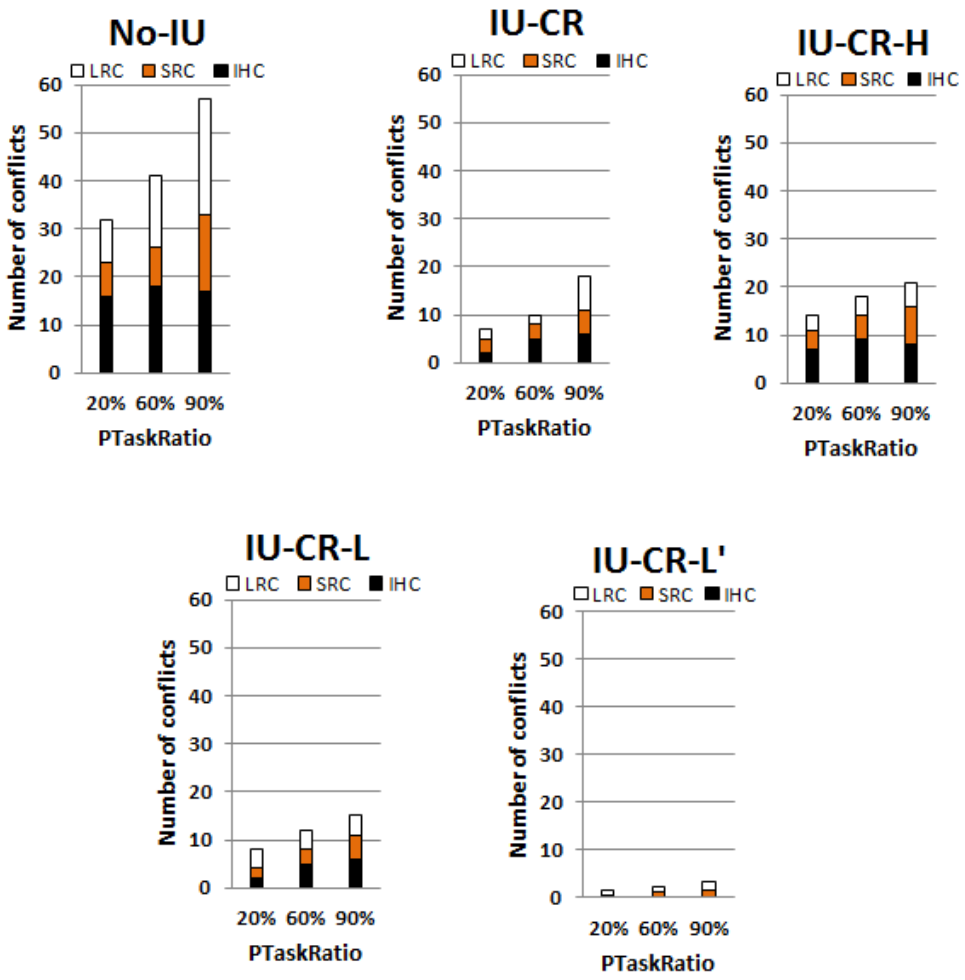


Figure 5.17: Number of conflicts (LRC, SRC and IHC) unresolved by the five algorithms for 12 MCCs, for *PTaskRatio* to be 20%, 60% and 90%.

In Figure 5.17, I observe that *IU-CR-L'* performs best on conflict resolution. *IU-CR-L'* reduces the number of conflicts unresolved by 96%, 94% and 94% for the 20%, 60% and 90% *PTaskRatio* cases respectively compared with *No-IU*. Max-sum is not guaranteed to resolve all the conflicts, especially when the networks are highly constrained.

Table 5.4 shows that *IU-CR-L'* expands the fewest number of states during online learning. *IU-CR-L'* unrolls significantly fewer states compared with *IU-CR*, *IU-CR-H* and *IU-CR-L* (reducing 97.6%, 46.4% and 22.7% respectively). Among these

Table 5.4: Comparison results after 10000 training episodes.

Approach	# of states expanded	# of *special states* expanded	# of newly learned sub-policies	Time spent in *MMLC* (sec.)
IU-CR	382765303	1547	1547	0.49
IU-CR-H	17432685	1084	1084	1.73
IU-CR-L	12097539	862	862	1.75
IU-CR-L'	9351376	621	621	2.38

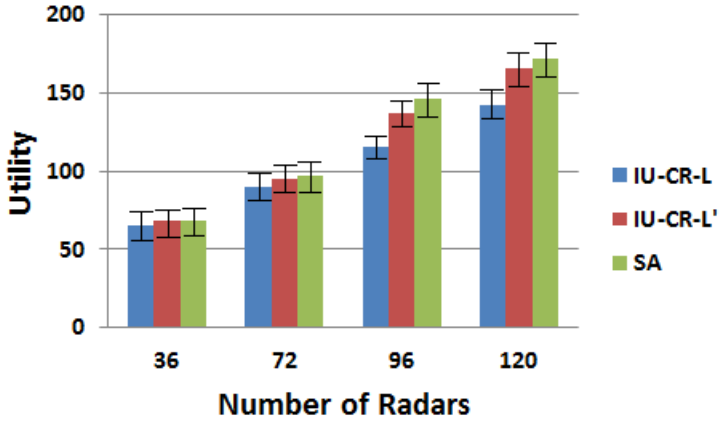
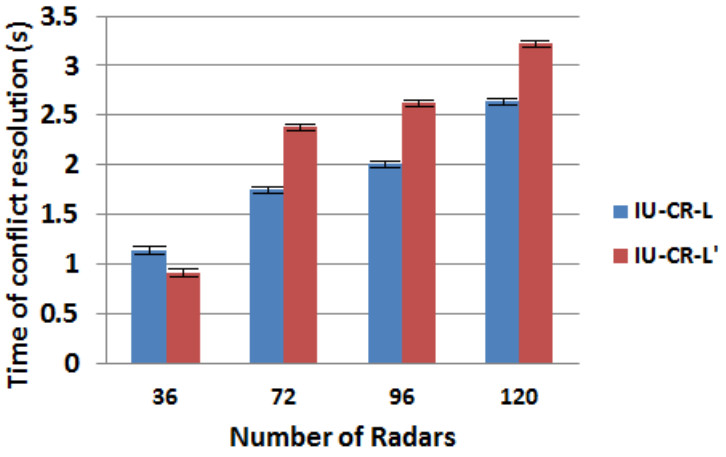


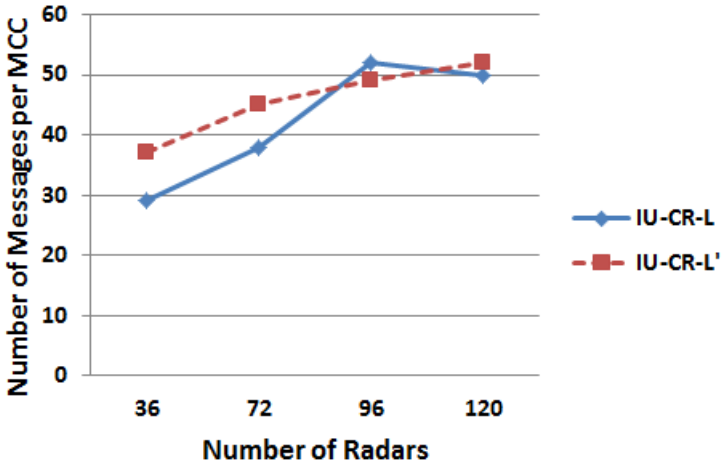
Figure 5.18: *Utility* of the three algorithms for 12 MCCs. The *SA* algorithm is run with the same number of tasks (weather phenomena) as the number of radars. It is run with a computation time limit of 6 minutes. I set the time limit to 6 minutes in order to get reasonable optimizations for sake of calculating the upper bound on *Utility*. I use the upper bound to measure how far the other two algorithms are from the global optimization solution.

expanded states only a very small fraction ($< 0.01\%$) are visited during learning. Each learned sub-policy associated with an encountered *special state* is iterated 503 times on average. The good performance of *IU-CR-L'* with regard to conflict resolution helps reduce the possibility of introducing more *special states* for agents, thus mitigating extra unrolling on search space. Although *IU-CR-L'* learns better performing policies compared to other algorithms, it spends more time in the *MMLC* phase (2.38 *sec.* compared with 0.49, 1.73 and 1.75 *sec.* respectively). This is because it takes longer time for the Max-sum algorithm to converge.

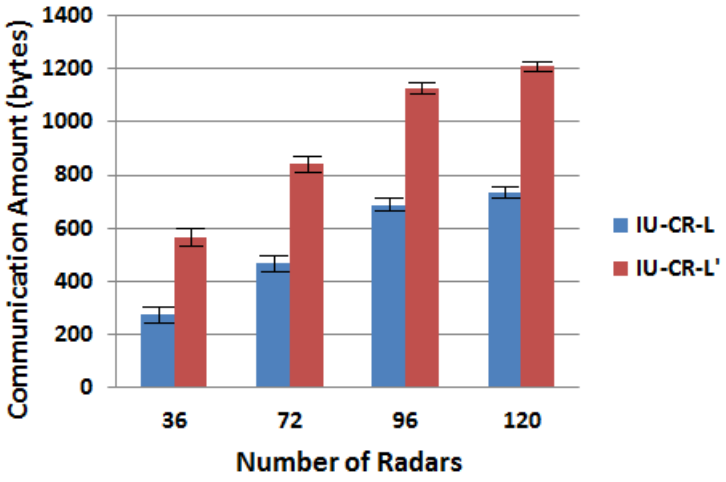
Different Network Sizes



(a) *Time in the decentralized setting*



(b) *Messages*



(c) *Communication*

Figure 5.19: Comparison of the decentralized negotiation algorithm in *IU-CR-L* and the Max-sum algorithm in *IU-CR-L'*.

In order to evaluate the general performance and the scalability of the algorithms, I compare the performance on different sized networks. In these scenarios, there are the same number of phenomena as the number of radars in the network as shown in Figure 5.18 and Figure 5.19. I used a simulated annealing (SA) [Kirkpatrick et al., 1983] algorithm to calculate globally optimal solutions and compared the learning approaches with this benchmark as upper bound. The *SA* algorithm approaches the global optimum by gradually reducing the intensity of making "downhill" moves.

It is observed in Figure 5.18 that *IU-CR-L'* performs better than *IU-CR-L* on *Utility* for the 36, 72, 96 and 120 radar cases respectively. The improvements are 5%, 6%, 19% and 17% respectively. *IU-CR-L'* remains quite close to the optimal solution with lower number of radars. *IU-CR-L'* achieves 99.7% and 98.2% on *Utility* compared with the *SA* algorithm with respect to 36 and 72 radar cases. Even for the 96 and 120 radar cases, *IU-CR-L'* achieves 93.8% and 96.5% on *Utility* compared to the upper bound. In some circumstances, the Max-sum algorithm is not guaranteed to converge and approximate solution is made that is deviated from the optimal one. *IU-CR-L* achieves 95.3%, 93.1%, 78.8% and 82.6% respectively on *Utility* compared with the upper bound.

The result in Figure 5.19 (a) shows that the Max-sum algorithm is able to handle the problem well both with respect to quality and computation time on bigger sized problems. The Max-sum algorithm achieves higher *Utility* at the cost of more computation time and communication cost. Compared to the Dec-Neg-MMLC algorithm, Max-sum spends 36%, 31% and 22% more time in the 72, 96 and 120 radar cases respectively. Max-sum needs more cycles for message transition and computation which takes more time. In the 36 radar case, Max-sum spends 19% less time compared to the other. In the network topology with lower constraints, the depth of the tree is lower which makes the message propagation more efficient thus reducing the convergence time of Max-sum. In terms of communication, when only messages exchanged

across MCCs are counted, Max-sum needs no more than twice the communication than the decentralized negotiation algorithm. Communication only between MCCs are measured for both algorithms.

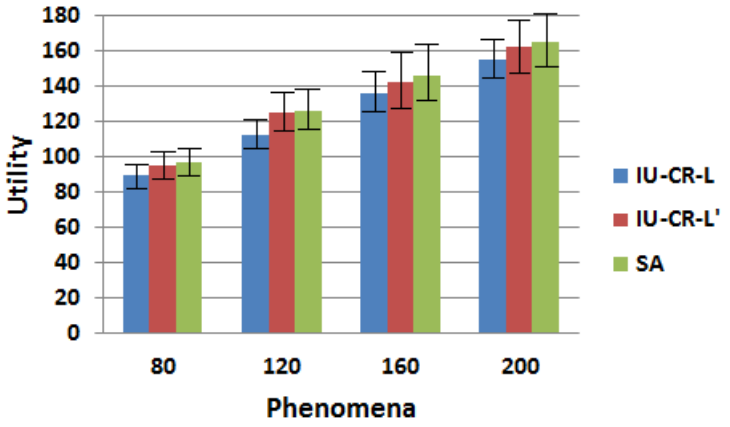
Different number of phenomena

In the next experiment, I increase the number of weather phenomena in a 72-radar network with 12 MCCs, thereby requiring more coordination among radars and study how the algorithms perform. Compared with the upper bound determined by SA, *IU-CR-L'* achieves 98%, 99%, 97% and 98% on *Utility* in the 80, 120, 160 and 200 phenomena cases respectively (Figure 5.20 (a)). While the quality of solution of Max-sum is slightly better, the time complexity of Max-sum sharply increases because in Max-sum more cycles are needed to calculate the optimal solution as more weather phenomena are added. *IU-CR-L'* increases 6%, 12%, 4% and 5% on *Utility*, while spends 36%, 40%, 49% and 61% more time in the four cases respectively.

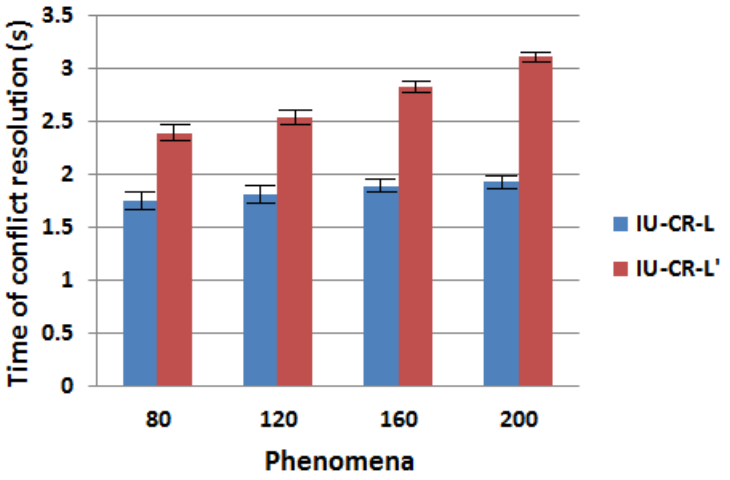
Also, the number of messages across MCCs increases as shown in Figure 5.20 (c) as there are more tasks shared by multiple MCCs in the environment. The number of messages in the decentralized negotiation algorithm increases slightly because the algorithm quickly converges to a suboptimal solution due to the termination condition inside each MCC resulting in an early termination within only 1 to 2 cycles.

Scaling up number of agents

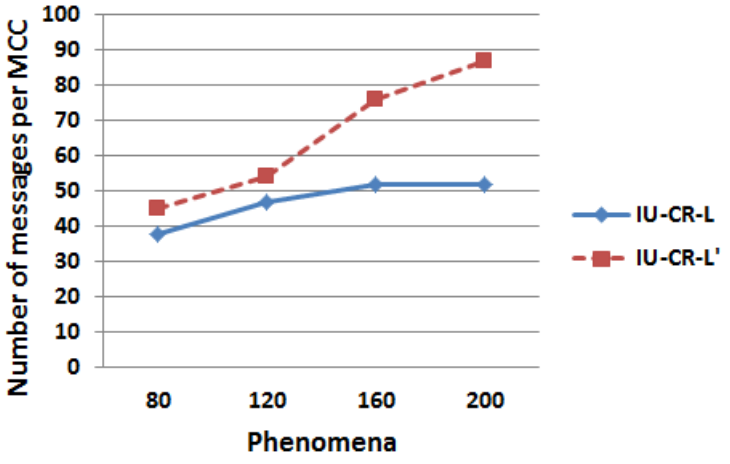
In the next experiment, I increase the number of MCCs to be 12, 18 and 30 (with 72, 108 and 180 radars respectively). Figure 5.21 (a) shows that *IU-CR-L'* outperforms (increases 6.3% and 10.4%) *IU-CR-L* on *Utility* in the 12 and 18 MCCs cases respectively. However, in the 30 MCCs cases, the decentralized negotiation algorithm performs better than Max-sum (with an increase of 8.4%) on *Utility*. The number of neighbor in function nodes in Max-sum increases as more agents are added that makes the factor graph not cycle free. Thus Max-sum does not converge during the *MMLC* phase (within the 6 second limit) and the optimal solution is not successfully



(a) *Utility*

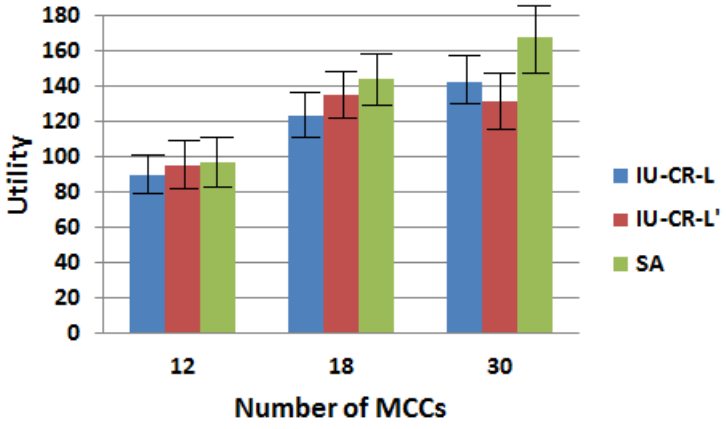


(b) *Time in the decentralized setting*

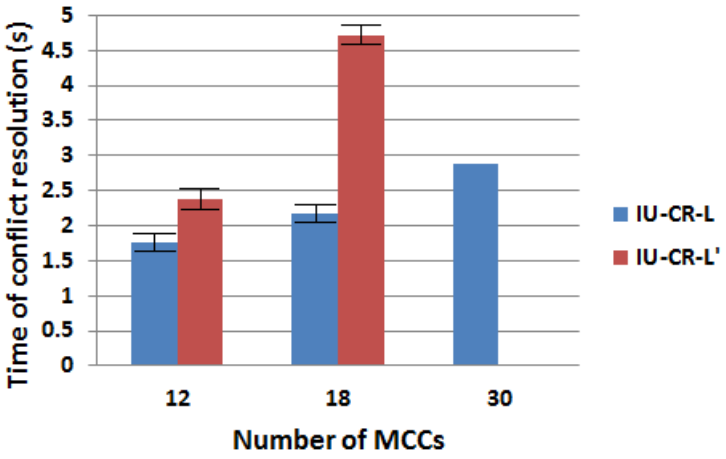


(c) *Messages*

Figure 5.20: Comparison of the decentralized negotiation algorithm in *IU-CR-L* and the Max-sum algorithm in *IU-CR-L'*, with 80, 120, 160 and 200 phenomena respectively.



(a) *Utility*



(b) *Time in the decentralized setting*

Figure 5.21: Comparison of the decentralized negotiation algorithm in *IU-CR-L* and the Max-sum algorithm in *IU-CR-L'*, with 12, 18 and 30 agents respectively.

generated. Even in such cases, Max-sum still generates good solutions (78.0% of the upper bound as in Figure 5.21 (a)). As Figure 5.21 (b) shows, the time complexity of Max-sum sharply increases when the number of agents increases from 12 to 18. In the 18 MCCs cases, Max-sum uses more than two times the time as the decentralized negotiation algorithm. I were not able to compute the time Max-sum uses in the 30 MCCs cases, since Max-sum did not converge in such cases within the 6 second limit.

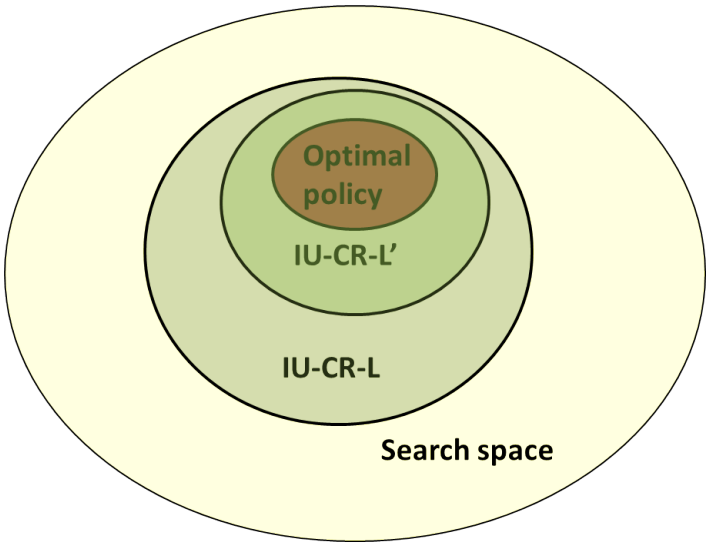


Figure 5.22: For smaller to medium sized networks (number of agents < 30), *IU-CR-L'* works on a more accurate search space than *IU-CR-L*.

5.4.3 Summary

There are compelling insights that resulted from extensive empirical evaluation of the *IU-CR-L'* algorithm with respect to the *IU-CR-L* algorithm. In all the evaluated scenarios with 12 and 18 agents, *IU-CR-L'* performs significantly better than *IU-CR-L* with respect to conflict resolution and utility. Although *IU-CR-L'* unrolls fewer states compared with the other algorithms, the improved performance with respect to conflict resolution helps reduce the possibility to introduce more *special states* for agents, thus mitigating extra unrolling on search space (as Figure 5.22 shows). In more dynamic scenarios (with more radars and more phenomena), *IU-CR-L'* outperforms *IU-CR-L* while the computation time and communication cost tend to increase substantially. As far as Max-sum converges within the *MMLC* phase, it generates optimal solutions.

In the empirical evaluation of scenarios with 30 agents, *IU-CR-L'* does not perform as well as *IU-CR-L* on utility. This is because in such cases, the factor graph of Max-sum is densely clustered and Max-sum is not converged within the *MMLC* phase. Approximate solutions may lead to biased subspace unrolling (where biased subspace

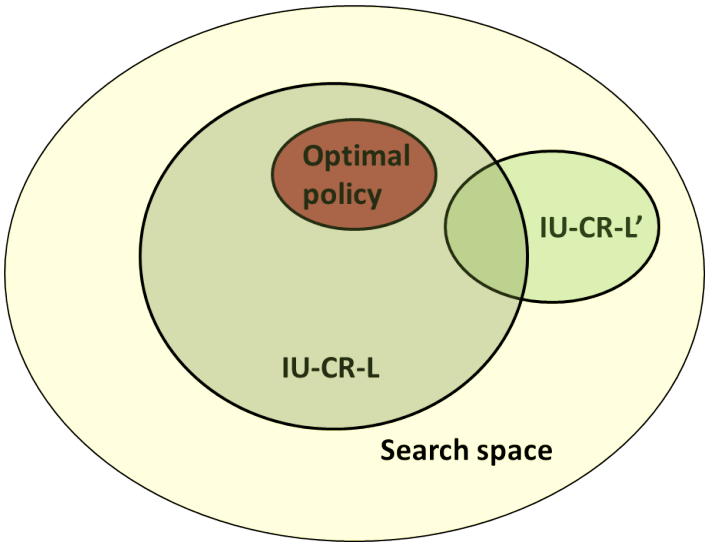


Figure 5.23: For bigger agent networks (number of agents ≥ 30), *IU-CR-L'* works on a biased search space.

means that if there are many equally likely outcomes for an action choice, the agent unrolls one subtree that has lower probability to reach, as Figure 5.23 shows).

This chapter describes the different types of conflicts that may exist in the problem domain and presents an adaptive approach to resolve conflicts that can arise between locally optimal policies. An algorithm that uses the heuristic rules to locally resolve simple conflicts is proposed first. When the environment becomes more dynamic and uncertain, the optimal policies are not easily learned using this algorithm. A decentralized learning approach is then presented to handle such complex environments. The approach resolves more complicated conflicts using a decentralized negotiation algorithm and selectively expands the agents' state space during the learning process. Experimental results show that this approach achieves good performance on system utility and conflict resolution by unrolling a small fraction of the whole search space. In order to resolve conflicts globally and reach the upper bound of globally optimal solution, I propose a decentralized learning approach that a DCOP algorithm is embedded to resolve conflicts and further pilot the state expansion. In most evaluated scenarios, the DCOP algorithm performs significantly better than the decentralized

negotiation algorithm on conflict resolution as well as on overall utility, with the price of message and communication overhead. However, in scenarios that have large number of agents (30 in NetRads domain), the DCOP algorithm is inferior to the decentralized negotiation algorithm with regard to utility.

CHAPTER 6: CONCLUSIONS

Decision making in cooperative multiagent systems is an important topic since many large-scale applications are formulated in terms of spatially or functionally distributed agents. Collaboration enables the different agents to work more efficiently and to complete activities they are not able to accomplish individually. However, in order to collaborate the agents should (learn to) coordinate their actions. This is a complicated process because the state space grows exponentially with an increase of the number of agents, and each agent takes individual decisions of which the outcome can be influenced by the actions performed by the other agents. Moreover, conflicts among the learned polices of individual agents could happen that may have detrimental influence on the overall performance.

This dissertation presented several techniques to coordinate and learn the behavior of the agents in distributed cooperative multiagent systems. It both studied the problem of coordinating the behavior of multiple agents in a specific situation, and learning, based on experience and conflict resolution performance, the behavior of a group of agents in sequential decision-making problems. The latter are problems in which the agents repeatedly interact with their environment and have to perform a sequence of actions in order to reach a certain goal. My main approach in all presented methods is to facilitate the learning problem by exploiting the on-line nonlocal state information in case of conflicts.

This dissertation establishes the following hypothesis: *Leveraging decentralized learning and conflict resolution helps converge to polices among complex agents that improve the overall performance of a cooperative multiagent system.* This final chapter presents several concluding remarks on the work described in this dissertation and

highlights its contributions. Furthermore, it discusses several promising directions for future research.

6.1 Main Results

A DEC-MDP framework that approximates the DEC-POMDP to learn coordinated joint policies for complex agents is presented in Chapter 3. The policies allow each agent to adapt to changes in environmental conditions while reorganizing the underlying multiagent network when needed. The exploration costs of DEC-MDPs are substantially decreased by constructing abstract classes of scenarios, states and actions. The abstract action is used during the learning stage while the detailed action is used at execution. The agents learn stochastic policies and approximate the solution to the DEC-MDP by using a factored reward function that captures the value of tasks from a partially global perspective instead of a local perspective.

Chapter 4 provides insight into the usefulness of reinforcement learning algorithms in complex multiagent sequential decision-making problems. I implemented a multiagent version of a RL-based algorithm, called PGA-APP, to learn the polices of the DEC-MDPs. This approach learns offline polices with a simplifying assumption that the entire network experiences one particular environmental scenario. A control flow framework in each agent is presented which controls the agent interaction and policy learning process. The learning is sped up by categorizing the real-world scenarios into different classes and learning policies separately for each class through controlled experimentation. I empirically compared my approach with the other two: the *No-MLC* that has no explicit or implicit meta-level control; and the *AHH* that incorporates hand-generated heuristics to make action choices. Experimental results show that my approach significantly outperforms ($p < 0.05$) *No-MLC* and *AHH* for different weather scenarios with different number of agents (3, 12 and 30 respectively). Results also show that my approach is better at handling dynamic environments (with more tasks and dependencies among agents).

Chapter 5 investigates the two related decentralized learning research questions namely, a) how to include critical contextual information when there is a very large search space for each agent? and b) how to resolve conflicts among the learned policies of different agents? I described the different types of conflicts that may exist in the problem domain and defined heuristic rules to locally resolve conflicts in simplified environments. I observed that heuristic rules are capable of resolving high percentage of conflicts in best cases, while in worst cases they are not resolving conflicts effectively.

I then presented a decentralized learning approach, called *IU-CR-L*, to handle more complex and dynamic environments. *IU-CR-L* takes advantage of offline policies within the context of a simplified environment and modifies these policies online based on experience gained in real environments. It learns (sub-)optimal polices for each agent by harnessing informed unrolling of state space and conflict resolution methods. A decentralized negotiation algorithm that builds on mediator mechanism to solve conflicts from a partially global perspective is proposed. Using *IU-CR-L* the agents were able to learn useful policies with a small amount of training (10000 episodes). *IU-CR-L* achieved significantly better performance on utility and conflict resolution by unrolling a small fraction (only 10% in the best cases) of the whole search space. I mapped the conflict resolution coordination problem as a DCOP formulation and replaced the previous negotiation algorithm with a DCOP algorithm, called Max-sum, to resolve complex conflicts from a global perspective so as to produce globally optimal solution for each agent. Experimental results show that the policies learned with Max-sum perform as well as if not better than the decentralized negotiation algorithm. In most scenarios, the learned policies using Max-sum achieved more than 90% on utility compared with the globally optimal solution (upper bound), while unrolling significantly ($p < 0.05$) less number of states. One useful result was that the Max-sum algorithm helped the agents unroll fewer number of states without losing

utility on performance.

6.2 Applying this Work

While this study focuses on meta-level questions in NetRads, I have framed the research questions to be applicable to the deliberative level as well. My approach can be applied to most MAS applications where information about its context is accessible and improves the agents' decision making performance and makes the MAS more coordinated. My approach also benefits from the assumptions that the number of *special states* that are added via online learning are limited; and that the learned policy has a finite horizon. Conflict resolution between agent policies can be handled explicitly (as is the case for NetRads in this research) or implicitly (in domains where conflicts lead to some reduction in utility).

This dissertation shows that my approach can be effective in real-time environments, characterized by uncertainty and limited computational resources. In these environments, computational resources such as time, memory or information are limited for policy learning and calculation. It shows that my approach is a flexible, real-time approach which seeks to optimize solution quality. The learning approach described in this dissertation allows the agent to capture the really important non-local context information based on experience gained in real environments, and explore the "right" part of the whole search space for globally optimal solution.

6.3 Future Extensions

Approximate methods

Experimental results in Chapter 5 show that Max-sum does not perform well in some scenarios for reasons such as taking long time to converge and having too much communication overhead. I plan to use extended versions of Max-sum, called fast Max-sum [Ramchurn et al., 2010] and bounded Max-sum [Stranders et al., 2009], as the contingency plan. Fast max-sum [Ramchurn et al., 2010] extends Max-sum in two main ways. First, it reduces the number of states over which each factor has to

compute its solution. Second, only the nodes who's utility changes as a result of the addition need resend their Max-sum messages. It uses less messages and converges much faster than Max-sum. Bounded Max-sum provides bounded approximate solutions by removing cycles in the original constraint network and then using Max-sum to optimally solve the resulting tree structured constraint network. These two extended versions generate good approximate solutions. They are well suited for large scale distributed applications in which the optimality of the solution can be sacrificed in favor of computational and communication efficiency [Ramchurn et al., 2010].

Field study

An operational radar testbed [Krainin et al., 2007] [Zink et al., 2005] that is deployed in Oklahoma is used to observe severe weather events and compare the performance of radar scanning using different techniques. I am interested in taking some of the ideas I have studied in the NetRads simulator and testing them out in the testbed. A verification of my algorithm in a real system would be a big win in understanding my approach.

Currently the 4 radar testbed in Oklahoma has been dismantled. The existing radars are being refurbished and will be installed in the Dallas Fort Worth area sometime this summer although the siting process might result in some delays in getting all 4 nodes up by the end of the summer. And the longer-term plan for Dallas should have at least 8 radars running in the next few years with a high possibility of an even larger network in the next 3 or 4 years. I will upload my algorithms in the testbed and test them out in the near future (probably this summer).

MMLC trigger

In my current implementation of PGA-APP, the *MMLC* phase is triggered at every heartbeat. This is acceptable since I assume radar switching has no cost and running the heuristic rule-based algorithm costs very little time (this is verified by evaluation). When I add cost for radar switching, triggering the MMLC too frequently may not be

a good idea. Consider the situation that the same radar is switched to a new MCC at the first heartbeat and then it is switched back in the next heartbeat. This results in a big cost (overhead) on radar switching compared with the limited utility gained and thus should be avoided. It would be interesting to develop different strategies to dynamically trigger the MMLC phase based on the system performance so as to efficiently utilize the limited resources.

Comparing with existing work on RL

I use abstract actions and states in reinforcement learning algorithm. I will compare my approach with other RL algorithms that also use a certain kind of abstraction in states and actions. It would also be interesting to compare my approach with other extended versions of RL (e.g., hierarchical RL).

REFERENCES

Abdallah, S. and Lesser, V. (2006). 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. ACM Press.

Alexander, G., Raja, A., Durfee, E., and Musliner, D. (2007). Design paradigms for meta-control in multi-agent systems. proceedings of aamas 2007 workshop on metareasoning in agent-based systems. pages 92–103, Hawaii.

Alexander, G., Raja, A., and Musliner, D. (2008). Controlling deliberation in a markov decision process-based agent. proceedings of the seventh international joint conference on autonomous agents and multi-agent systems (aamas08). pages 461–468, Estoril, Portugal.

An, B., Lesser, V., Westbrook, D., and Zink, M. (2011). Agent-mediated Multi-step Optimization for Resource Allocation in Distributed Sensor Networks. Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems Innovative Applications Track (AAMAS 2011). Taipei, Taiwan. IFAAMAS.

Banerjee, B. and Peng, J. (2007). Generalized multiagent learning with performance bound. autonomous agents and multi-agent systems. 15(3):281–312.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. artif. intell. 72(1-2):81–138.

Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1989). Sequential decision probelms and neural networks. nips. pages 686–693.

Becker, R., Zilberstein, S., Lesser, V. R., and Goldman, C. V. (2003). Transition-independent decentralized markov decision processes. aamas. pages 41–48.

Becker, R., Zilberstein, S., Lesser, V. R., and Goldman, C. V. (2004). Solving transition independent decentralized markov decision processes. j. artif. intell. res. (jair). 22:423–455.

Bellman, R. E. (1957). Dynamic programming. Princeton University Press.

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

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. math. oper. res. 27(4):819–840.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming. Athena Scientific, Belmont, MA.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. ijcai. pages 478–485.

Bowling, M. and Veloso, M. (2002a). Multiagent learning using a variable learning rate. artificial intelligence. 136:215–250.

Bowling, M. and Veloso, M. (2002b). Scalable Learning in Stochastic Games. Proceedings of AAAI 2002 Workshop on Game Theoretic and Decision Theoretic Agents.

Carlin, A. and Zilberstein, S. (2011). Decentralized monitoring of distributed anytime algorithms. aamas.

Chalkiadakis, G., Robu, V., Kota, R., Rogers, A., and Jennings, N. R. (2011). Cooperatives of distributed energy resources for efficient virtual power plants. aamas. pages 787–794.

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

Cheng, S., Raja, A., and Lesser, V. R. (2010b). Multiagent meta-level control for predicting meteorological phenomena. proceedings of aaai-2010 workshop on metareasoning in robust social systems. pages 6–13, Atlanta, GA.

Cheng, S., Raja, A., and Lesser, V. R. (2010c). Towards multiagent meta-level control. aaai.

Conitzer, V. and Sandholm, T. (2007). 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.

Cox, M. and Raja, A. (2008). Metareasoning: A Manifesto. Proceedings of AAAI 2008 Workshop on Metareasoning: Thinking about Thinking. pages 1–4, Chicago,IL.

Cox, M. and Raja, A. (2011). Chapter: Metareasoning: An introduction. metareasoning: Thinking about thinking. MIT Press.

Cox, M. T. (2005). Metacognition in computation: A selected research review. artif. intell. 169(2):104–141.

Crites, R. H. and Barto, A. G. (1995). Improving elevator performance using reinforcement learning. nips. pages 1017–1023.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. E. (1995). Planning under time constraints in stochastic domains. artif. intell. 76(1-2):35–74.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. j. artif. intell. res. (jair). 13:227–303.

Doyle, J. (1983). What is rational psychology? toward a modern mental philosophy. ai magazine. 4(3):50–53.

Dutta, P. S., Jennings, N. R., and Moreau, L. (2005). Cooperative information sharing to improve distributed learning in multi-agent systems. j. artif. intell. res. (jair). 24:407–463.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. (2008). Decentralised Coordination of Low-power Embedded Devices Using the Max-sum Algorithm. The Seventh International Conference on Autonomous Agents and Multiagent Systems. pages 639–646, Estoril, Portugal.

Ghavamzadeh, M. and Mahadevan, S. (2004). Learning to communicate and act using hierarchical reinforcement learning. aamas. pages 1114–1121.

Goldman, C. V. and Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. j. artif. intell. res. (jair). 22:143–174.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored mdps. nips. pages 1523–1530.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. icml. pages 227–234.

Horvitz, E. (1988). Reasoning under varying and uncertain resource constraints. aaai. pages 111–116.

Howard, R. A. (1960). Dynamic programming and Markov Processes. The MIT Press.

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. icml. pages 242–250.

Hu, J. and Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. journal of machine learning research. 4:1039–1069.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. neural computation. 6(6):1185–1201.

Jain, M., Taylor, M. E., Tambe, M., and Yokoo, M. (2009). Dcops meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. ijcai. pages 181–186.

Kennedy, C. (2010). Decentralised metacognition in context-aware autonomic systems: Some key challenges. workshops at the twenty-fourth aaai conference on artificial intelligence.

Kho, J., Long, T.-T., Rogers, A., and Jennings, N. R. (2009). Decentralised control of adaptive sampling and routing in wireless visual sensor networks. aamas (2). pages 1237–1238.

Kim, Y., Krainin, M., and Lesser, V. (2010). Application of Max-Sum Algorithm to Radar Coordination and Scheduling. Twelfth International Workshop on Distributed Constraint Reasoning. pages 5–19, Toronto.

Kirkpatrick, S., Jr., D. G., and Vecchi, M. P. (1983). Optimization by simmulated annealing. science. 220(4598):671–680.

Kok, J. R. and Vlassis, N. A. (2006). Collaborative multiagent reinforcement learning by payoff propagation. journal of machine learning research. 7:1789–1828.

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

Kurose, J., Lyons, E., McLaughlin, D., Pepyne, D., Philips, B., Westbrook, D., and Zink, M. (2006). An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. proceedings of the second asian internet engineering conference, aintec. pages 1–15.

Lesser, V. R. (2003). Experiences building a distributed sensor network, canadian conference on ai. pages 1–6.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. icml. pages 157–163.

Low, K. H., Dolan, J. M., and Khosla, P. K. (2011). Active markov information-theoretic path planning for robotic environmental sensing. aamas. pages 753–760.

Mailler, R. and Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. proc. third international joint conference on autonomous agents and multiagent systems (aamas 2004). volume 1, pages 438–445.

Mailler, R. and Lesser, V. R. (2006). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. j. artif. intell. res. (jair). 25:529–576.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. icml. pages 361–368.

Modi, P., Scerri, P., Shen, W.-M., and Tambe, M. (2003). Distributed sensor networks: A multiagent perspective. kluwer academic publishers.

Modi, P. J., Shen, W., Tambe, M., and Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. ai journal. volume 161, pages 149–180.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. machine learning. 13:103–130.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. aaai. pages 133–139.

Nash, J. F. (1950). Equilibrium points in n-person games. proceedings of the national academy of science. pages 48–49.

Osborne, M. J. and Rubinstein, A. (1994). A course in game theory. The MIT Press.

Peng, J. and Williams, R. J. (1996). Incremental multi-step q-learning. machine learning. 22(1-3):283–290.

Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). Learning to cooperate via policy search. uai. pages 489–496.

Petcu, A. and Faltings, B. (2005a). DPOP: A scalable method for multiagent constraint optimization. proceedings of international joint conference on artificial intelligence (ijcai). pages 266–271.

Petcu, A. and Faltings, B. (2005b). S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. proceedings of the national conference on artificial intelligence (aaai-05). pages 449–454, Pittsburgh, Pennsylvania.

Petcu, A., Faltings, B., and Mailler, R. (2007). Pc-dpop: A new partial centralization algorithm for distributed optimization. ijcai. pages 167–172.

Petcu, A., Faltings, B., and Parkes, D. C. (2008). M-dpop: Faithful distributed implementation of efficient social choice problems. j. artif. intell. res. (jair). 32:705–755.

Puterman, M. L. (1994). Markov decision processes: Discrete stochastic dynamic programming. John Wiley Sons, Inc.

Pynadath, D. V. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. j. artif. intell. res. (jair). 16:389–423.

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

Raja, A. and Lesser, V. R. (2003). Efficient meta-level control in bounded-rational agents. aamas. pages 1104–1105.

Ramchurn, S. D., Farinelli, A., Macarthur, K. S., and Jennings, N. R. (2010). Decentralized coordination in robocup rescue. comput. j. 53(9):1447–1461.

Russell, S. and Norvig, P. (2003). Artificial intelligence: A modern approach. prentice hall, 2nd edition.

Russell, S. J. and Norvig, P. (2006). Artificial Intelligence A Modern Approach. Pearson Education.

Russell, S. J., Subramanian, D., and Parr, R. (1993). Provably bounded optimal agents. ijcai. pages 338–345.

Schiex, T., Fargier, H., and Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. ijcai (1). pages 631–639.

Schut, M. C. and Wooldridge, M. (2001). Principles of intention reconsideration. agents. pages 340–347.

Shafi, K. and Merrick, K. E. (2011). A curious agent for network anomaly detection. aamas. pages 1075–1076.

Shapley, L. (1953). Stochastic games. proceedings of the national academy of sciences. 39:1095–1100.

Simon, H. (1976). From substantive to procedural rationality. In S.J.Latsis (Ed.), Method and appraisal in economics. pages 129–148, Cambridge, UK. Cambridge University Press.

Simon, H. A. (1982). Models of Bounded Rationality. The MIT Press, Cambridge, Massachusetts.

Singh, S. P., Kearns, M. J., and Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. proceedings of uncertainty in artificial intelligence. pages 541–548.

Stranders, R., Farinelli, A., Rogers, A., and Jennings, N. (2009). Bounded approximate decentralised coordination using the max-sum algorithm. proceedings of 21st int. joint conference on ai(ijcai) 2009. Pasadena, USA.

Stranjak, A., Dutta, P. S., Ebden, M., Rogers, A., and Vytelingum, P. (2008). A multi-agent simulation system for prediction and scheduling of aero engine overhaul. aamas (industry track). pages 81–88.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. machine learning. 3:9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. ml. pages 216–224.

Sutton, R. S. and Barto, A. G. (1998). Reinforcement Learning. MIT Press.

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

Sycara, K. P. (1998). Multiagent systems, ai magazine. 19(2):79–92.

Tesauro, G. (1992). Practical issues in temporal difference learning. machine learning. 8:257–277.

Watkins, C. J. C. H. and Dayan, P. (1992). Technical note q-learning. machine learning. 8:279–292.

Weiss, G. (1999). Multiagent systems: A modern approach to distributed artificial intelligence. The MIT Press.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act by trial and error. machine learning. 7:45–83.

Wooldridge, M. (2002). An introduction to multiagent systems. john wiley & sons, ltd.,.

Wu, J. and Durfee, E. H. (2007). Solving large TAEMS problems efficiently by selective exploration and decomposition. proceedings of the sixth international conference on autononous agents and multi-agent ssytems (aamas 07). page 56.

Xuan, P. and Lesser, V. R. (2002). Multi-agent policies: from centralized ones to decentralized ones. aamas. pages 1098–1105.

Zhang, C., Abdallah, S., and Lesser, V. (2009). Integrating Organizational Control into Multi-Agent Learning. Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems. Budapest, Hungary.

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

Zhang, C. and Lesser, V. R. (2011). Coordinated multi-agent reinforcement learning in networked distributed pomdps. proceedings of the 25th national conference on argificial intelligence (aaai).

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

Zhang, W. and Wittenburg, L. (2003). Distributed breakout algorithm for distributed constraint optimization problems - DBArelax. proceedings of international joint conference on autonomous agents and multi agent systems (aamas).

Zhang, X., Lesser, V. R., and Abdallah, S. (2005b). Efficient management of multi-linked negotiation based on a formalized model. autonomous agents and multi-agent systems. 10(2):165–205.

Zink, M., Westbrook, D., Abdallah, S., Horling, B., Lyons, E., Lakamraju, V., Manfredi, V., Kurose, J., and Hondl, K. (2005). Meteorological Command and Control: An End-to-end Architecture for a Hazardous Weather Detection Sensor Network.

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

APPENDIX A: SYSTEM LOG FOR MOTIVATING EXAMPLE

Direct Application of Offline Policy

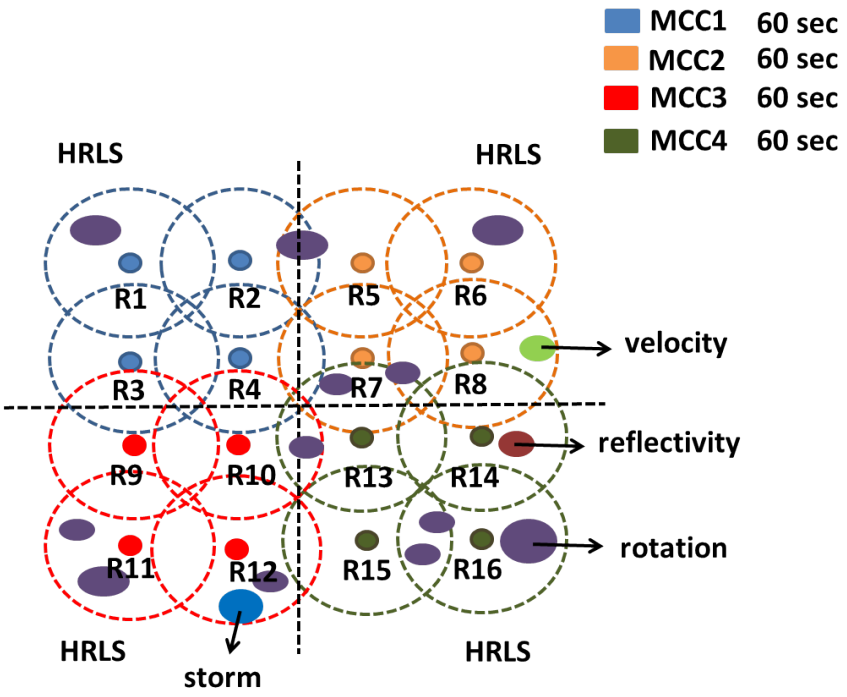


Figure A.1: The $MCCs$ experience homogeneous weather scenario.

I use an example to elaborate my *IU-CR-L* algorithm. Figure A.1 is a simple online scenario, where MCC_1 and all its neighbors experience the same weather scenario: HRLS. All the MCCs have the 60 second heartbeat. MCC_1, MCC_3 and MCC_4 each have one radar involved in the data correlation while MCC_2 has three radars involved in the data correlation. There is only one pinpointing task which appears in the overlapping area between R_2 and R_5.

In my approach, each MCC initially unrolls a small portion of its complete MDP space (line 1-3, Algorithm 3), called the initial MDP space S^{init}. For each state $s \in S^{init}$, the dominant action is unrolled using the offline policy computed for the current weather scenario. For all the subsequent states of this dominant action, the unrolling proceeds in a similar fashion until all the terminated states are reached. I

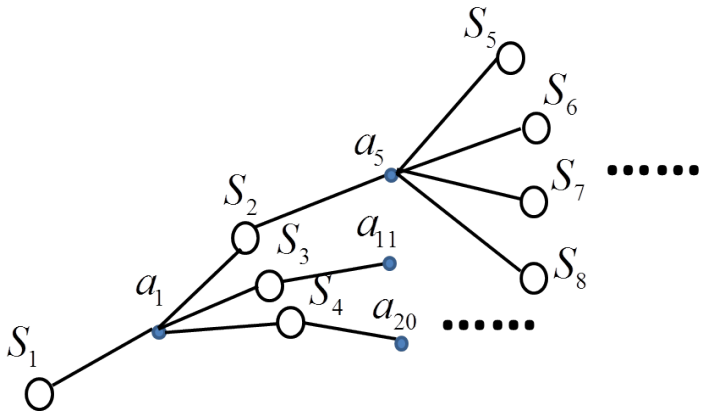


Figure A.2: S^{init} with initial state S_1 for MCC_1 for weather scenario: HRLS.

unroll S^{init} to balance the benefits of unrolling more states and the costs of unrolling by using selective unrolling. Figure A.2 shows the S^{init} with initial state S_1.

At time 0 of the *MMLC* phase, MCC_1 determines the weather scenario (HRLS) using the number of weather tasks of each type in the agent's scanning region (line 6, Algorithm 3). MCC_1 communicates with each of its neighbors to collect each neighbor's information about *data correlation*, the number of radars involved in *data correlation* and about their current heartbeat (line 7, Algorithm 3). MCC_1 then uses such information to determine its state (S_1 in Fig A.2). MCC_2, MCC_3 and MCC_4 simultaneously determine their state in a similar fashion.

Time 0 sec (MMLC phase):

MCC_1's state S_1:

$F_0 : (HRLS, 60seconds, 1)$;

$F_1 : \langle(60seconds, many), (60seconds, 1), (60seconds, 1)\rangle$;

$F_2 : \langle High, Low, Low\rangle$; (The percentage of pinpointing tasks in the overlapping areas between MCC_1 and MCC_2 is 100%, so the degree of data correlation is *High*. Similarly, the percentage of pinpointing tasks in the overlapping areas between MCC_1 and MCC_3 (MCC_4) is 0%, so the degree of data correlation is *Low*.)

Policy: { 63% 'Light Move (MCC_1 to MCC_2)' 'Use 60 seconds heartbeat'; 23% 'Heavy Move (MCC_1 to MCC_3)' 'Use 60 seconds heartbeat'; 5% 'Light Move (MCC_1

to MCC_3)' 'Use 60 seconds heartbeat'; 3% 'Heavy Move (MCC_1 to MCC_4) & Light Move (MCC_1 to MCC_2)' 'Use 30 seconds heartbeat'; 3% 'Light Move (MCC_1 to MCC_4)' 'Use 30 seconds heartbeat'; 1% 'No Move' 'Use 60 seconds heartbeat'; 0%... } (It is the offline policy of MCC_1 stored in the *scenario library*. The actions that have the probability distribution of 0 are omitted here for the sake of space.)

Action choice a_1 (Fig A.2): 'Light Move (MCC_1 to MCC_2)' 'Use 60 seconds heartbeat'. (MCC_1 chooses its action to take according to its policy (line 8, Algorithm 3). It is possible that other actions except the dominant action are chosen. In this case, I unroll the selected action and its subsequent state space (line 9-10, Algorithm 3).)

MCC_2's state:

$F_0 : (HRLS, 60seconds, many)$;

$F_1 : \langle(60seconds, 1), (60seconds, 1), (60seconds, 1)\rangle$;

$F_2 : \langle High, Low, Low\rangle$; (The percentage of pinpointing tasks in the overlapping areas between MCC_2 and MCC_1 is 100%, so the degree of data correlation is *High*. Similarly, the percentage of pinpointing tasks in the overlapping areas between MCC_2 and MCC_3 (MCC_4) is 0%, so the degree of data correlation is *Low*.)

Policy: { 57% 'Light Move (MCC_2 to MCC_3)' 'Use 60 seconds heartbeat'; 31% 'Light Move (MCC_2 to MCC_3) & Light Move (MCC_2 to MCC_4)' 'Use 60 seconds heartbeat'; 6% 'Heavy Move (MCC_2 to MCC_4)' 'Use 30 seconds heartbeat'; ... } (It is the offline policy of MCC_2 stored in the *scenario library*.)

Action choice: 'Light Move (MCC_2 to MCC_3)' 'Use 60 seconds heartbeat'.

MCC_3's state:

$F_0 : (HRLS, 60seconds, 1)$;

$F_1 : \langle(60seconds, 1), (60seconds, many), (60seconds, 1)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

Policy: { 49% 'Light Move (MCC_3 to MCC_4)' 'Use 60 seconds heartbeat'; 36% 'Light Move (MCC_3 to MCC_1)' 'Use 60 seconds heartbeat'; 10% 'No Move ' 'Use

60 seconds heartbeat'; ... } (It is the offline policy of MCC_3 stored in the *scenario library.*)

Action choice: 'Light Move (MCC_3 to MCC_4)' 'Use 60 seconds heartbeat'.

MCC_4's state:

$F_0 : (HRLS, 60seconds, 1)$;

$F_1 : \langle(60seconds, 1), (60seconds, many), (60seconds, 1)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

Policy: { 61% 'Light Move (MCC_4 to MCC_1)' 'Use 60 seconds heartbeat'; 22% 'Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat'; 7% 'Light Move (MCC_4 to MCC_3)' 'Use 60 seconds heartbeat'; ... } (It is the offline policy of MCC_4 stored in the *scenario library.*)

Action choice: 'Light Move (MCC_4 to MCC_1)' 'Use 60 seconds heartbeat'.

Each MCC applies its detailed action[1] of the abstract action and computes the number of conflicts in its neighborhood (line 12, Algorithm 3). The detailed actions for each meta-level action of radar reorganization are:

MCC_1: 'Move R_2 from MCC_1 to MCC_2'(The detailed actions for 'Light Move (MCC_1 to MCC_2)' along with their frequency to be applied during offline learning is: $\emptyset$: 20%; 'Move R_1 from MCC_1 to MCC_2': 11%; 'Move R_2 from MCC_1 to MCC_2': 43%; 'Move R_3 from MCC_1 to MCC_2': 16%; 'Move R_4 from MCC_1 to MCC_2': 10%.)

MCC_2: $\emptyset$

MCC_3: 'Move R_{10} from MCC_3 to MCC_4'

MCC_4: $\emptyset$

I can see that no conflict occurs in this scenario, thus there is no need to resolve

[1]There could be different detailed actions for one abstract action. For instance, 'Light Move (MCC_1 to MCC_2)' can have such detailed actions as: 'Move R_1 from MCC_1 to MCC_2'; 'Move R_2 from MCC_1 to MCC_2' and $\emptyset$. $\emptyset$ means no radar needs to be moved and it could be one detailed action of 'Light Move'. In [Cheng et al., 2010a], each MCC applies the detailed action that is most likely to be applied based on the history of the training data. Such detailed actions capture the effect of policy best.

conflicts:

LRC: 0

SRC: 0

IHC: 0

Each MCC observes the next state s' (S_2 in Fig A.2) without adding overlapping context about its neighbors (line 15, Algorithm 3. The changed features are bolded, such changes are due to new action choices):

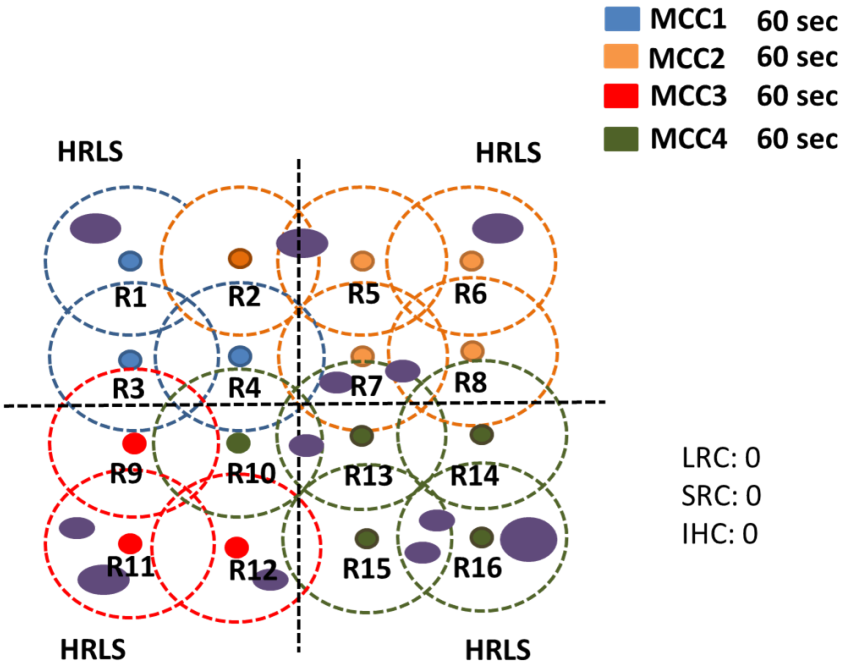


Figure A.3: MCC-Radar configuration for the homogeneous scenario at time 0.08 sec.

Time 0.08 sec (MMLC phase):

MCC_1's state S_2:

$F_0 : (HRLS, 60seconds, 0)$;

$F_1 : \langle(60seconds, many), (60seconds, 0), (60seconds, 1)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

MCC_2's state:

$F_0 : (HRLS, 60seconds, many)$;

$F_1 : \langle(60seconds, 0), (60seconds, 0), (60seconds, 1)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

MCC_3's state:

$F_0 : (HRLS, 60seconds, 0)$;

$F_1 : \langle(60seconds, 0), (60seconds, many), (60seconds, 1)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

MCC_4's state:

$F_0 : (HRLS, 60seconds, 1)$;

$F_1 : \langle(60seconds, 0), (60seconds, many), (60seconds, 0)\rangle$;

$F_2 : \langle Low, Low, Low\rangle$;

I can see that S_2 has already been explored in the S^{init} for MCC_1 (Figure A.2). This is also true for the other three $MCCs$. Because no conflict exists among the actions of each MCC, there is no need to execute the iterations of UMDP and CR stages (line 22, Algorithm 3) as described in Chapter 5.3. As defined in [Cheng et al., 2010a], the *termination condition* is met when the conflict resolution performance is good or the time limit for the $MMLC$ phase is reached. The state space of each MCC remains unchanged and there are no additional features added to the current state of any MCC.

Indirect Application of Offline Policy

Figure A.4 is a complex online scenario, where MCC_1 and its neighbors experience different weather scenarios. MCC_1, MCC_2, MCC_3 and MCC_4 experiences HRLS, HRMS, MRMS and LRHS respectively. All the $MCCs$ have 60 second heartbeat. MCC_1 has one radar involved in the data correlation while MCC_2, MCC_3 and MCC_4 each have three radars involved in the data correlation. Suppose MCC_1 and MCC_4 are almost overloaded[2], and moving two radars simultaneously to MCC_1 or MCC_4

[2]The load of each MCC is measured based on the amount of data it needs to process. The number of radar it controls as well as the number of tasks (especially pinpointing tasks) in the overlapping areas contribute to the load.

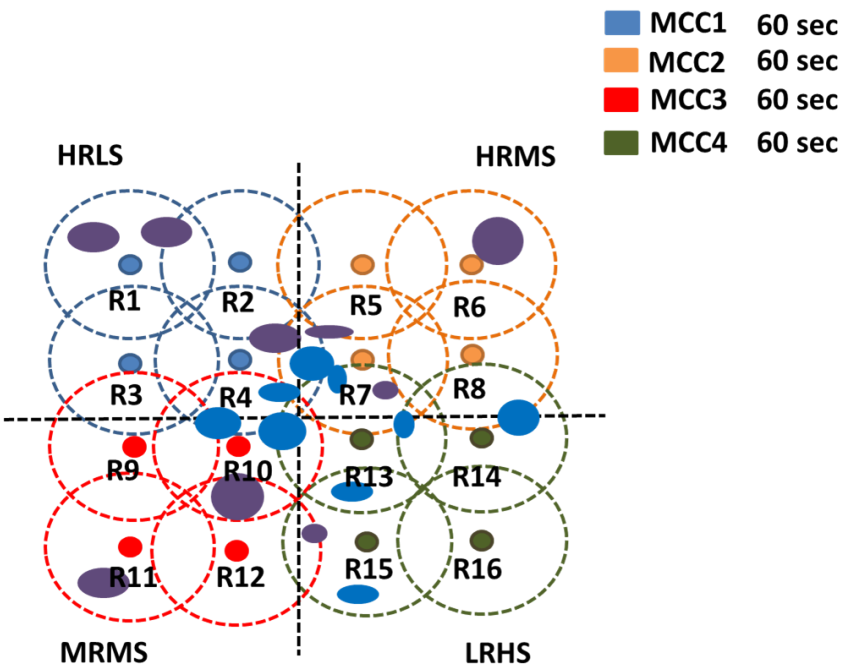


Figure A.4: The $MCCs$ experience heterogeneous weather scenarios.

results in an LRC. Direct application of the offline policy of MCC_1 could lead to undesirable performance since the offline policy does not have the current context of the agent. The tasks and data shared between MCC_1 and its neighbors are more complex than that in the simple online scenario. For example, 'Use 60 seconds heartbeat' is a good choice for MCC_1 in the simple online scenario where each MCC experiences the same weather scenario: HRLS. When MCC_1's neighbors experience the weather scenarios where the number of storms is predominant, adhering to the 60 seconds heartbeat is a bad choice that harms the overall performance. A shorter heartbeat allows MCCs to adapt to changing weather conditions (especially storms). This means MCC_1 requires non-local information from other agents in order to make globally relevant decisions. Additional conflicts between MCC_1 and its neighbors could occur.

As described in Chapter 5.3, my approach is that each MCC initially unrolls its S^{init} (line 1-3, Algorithm 3). Each MCC selectively expands the search space to

capture non-local context information crucial to global performance and to negotiate about conflicts that cannot be resolved locally.

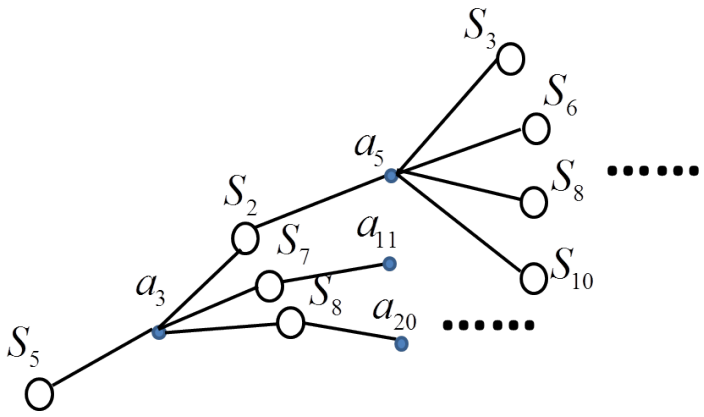


Figure A.5: S^{init} with initial state S_5 for MCC_1 for weather scenario: HRLS.

At time 0 sec ($MMLC$ phase), MCC_1 identifies that it is in the weather scenario: HRLS (line 6, Algorithm 3). MCC_1 communicates with each of its neighbors to collect their information about their *data correlation*, the number of radars involved in *data correlation* and their heartbeat. MCC_1 uses such information to calculate the features of its state correspondingly (line 7, Algorithm 3). MCC_2, MCC_3 and MCC_4 also simultaneously identify their weather scenario (HRMS, MRMS and LRHS respectively) and calculate the features of their state.

Time 0 sec (MMLC phase):

MCC_1's state S_5 (Fig A.5):

$F_0 : (HRLS, 60seconds, 1)$;

$F_1 : \langle(60seconds, many), (60seconds, many), (60seconds, many)\rangle$;

$F_2 : \langle High, Low, Medium\rangle$; (The percentage of pinpointing tasks between MCC_1 and MCC_2 is 75%. The percentage of pinpointing tasks between MCC_1 and MCC_3 is 0%. The percentage of pinpointing tasks between MCC_1 and MCC_4 is 33%.)

Policy: { 74% 'Light Move (MCC_1 to MCC_4)' 'Use 60 seconds heartbeat'; 12% 'Light Move (MCC_1 to MCC_2)' 'Use 60 seconds heartbeat'; 4% 'Heavy Move (MCC_1 to MCC_3)' 'Use 60 seconds heartbeat';... }

Action choice a_3: 'Light Move (MCC_1 to MCC_4)' 'Use 60 seconds heartbeat' (MCC_1 chooses its action to take according to its policy (line 8, Algorithm 3). It is possible that other actions except the dominant action are chosen. In this case, I unroll the selected action and its subsequent state space (line 9-10, Algorithm 3).)

MCC_2's state:

$F_0 : (HRMS, 60seconds, many)$;

$F_1 : \langle(60seconds, 1), (60seconds, many), (60seconds, many)\rangle$;

$F_2 : \langle High, Low, High\rangle$;

Policy: { 43% 'Heavy Move (MCC_2 to MCC_4)' 'Use 60 seconds heartbeat'; 30% 'Heavy Move (MCC_2 to MCC_1) & Heavy Move (MCC_2 to MCC_3)' 'Use 60 seconds heartbeat'; 16% 'Heavy Move (MCC_2 to MCC_3)' 'Use 60 seconds heartbeat';... }

Action choice: 'Heavy Move (MCC_2 to MCC_4)' 'Use 60 seconds heartbeat'

MCC_3's state:

$F_0 : (MRMS, 60seconds, many)$;

$F_1 : \langle(60seconds, 1), (60seconds, many), (60seconds, many)\rangle$;

$F_2 : \langle Low, Low, Medium\rangle$;

Policy: { 52% 'Light Move (MCC_3 to MCC_1)' 'Use 30 seconds heartbeat'; 23% 'Heavy Move (MCC_3 to MCC_2)' 'Use 60 seconds heartbeat'; 5% 'No Move' 'Use 30 seconds heartbeat';... }

Action choice: 'Light Move (MCC_3 to MCC_1)' 'Use 30 seconds heartbeat'

MCC_4's state:

$F_0 : (LRHS, 60seconds, many)$;

$F_1 : \langle(60seconds, 1), (60seconds, many), (60seconds, many)\rangle$;

$F_2 : \langle Medium, High, Medium\rangle$;

Policy: { 39% 'Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat'; 20% 'Heavy Move (MCC_4 to MCC_1) & Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat'; 13% 'Heavy Move (MCC_4 to MCC_2)' 'Use 60 seconds heartbeat';... }

Action choice: 'Heavy Move (MCC_4 to MCC_1) & Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat'

Figure A.5 shows the S^{init} with initial state S_5 for MCC_1. Each MCC applies its detailed action of the abstract action and computes the number of conflicts in its neighborhood (line 12, Algorithm 3). The detailed actions for each action of radar reorganization are:

MCC_1: 'Move R_4 from MCC_1 to MCC_4' (The detailed actions for 'Light Move (MCC_1 to MCC_4)' along with their frequency to be applied during offline learning is: $\emptyset$: 30%; 'Move R_1 from MCC_1 to MCC_4': 23%; 'Move R_2 from MCC_1 to MCC_4': 1%; 'Move R_3 from MCC_1 to MCC_4': 9%; 'Move R_4 from MCC_1 to MCC_4': 37%.)

MCC_2: 'Move R_5 and R_6 from MCC_2 to MCC_4'

MCC_3: 'Move R_{10} from MCC_3 to MCC_1'

MCC_4: 'Move R_{13} and R_{14} from MCC_4 to MCC_1' 'Move R_{13}, R_{14} and R_{15} from MCC_4 to MCC_2'

The initial numbers of all the types of conflicts are:

LRC: 2 (LRC_1: MCC_1 and MCC_2 move radars simultaneously to MCC_4; LRC_2: MCC_3 and MCC_4 move radars simultaneously to MCC_1.)

SRC: 2 (SRC_1: MCC_1 and MCC_2 both want the control of R_{13}; SRC_2: MCC_1 and MCC_2 both want the control of R_{14}.)

IHC: 4 (IHC_1: MCC_1 and MCC_3 use different heartbeats; IHC_2: MCC_1 and MCC_4 use different heartbeats; IHC_3: MCC_2 and MCC_3 use different heartbeats; IHC_4: MCC_2 and MCC_4 use different heartbeats.)

The MCCs use the decentralized negotiation algorithm to resolve conflicts (line 13, Algorithm 3), the detailed actions of all MCCs are updated as:

Time 0.44 sec (MMLC phase):

Round 1 (UMDP and CR):

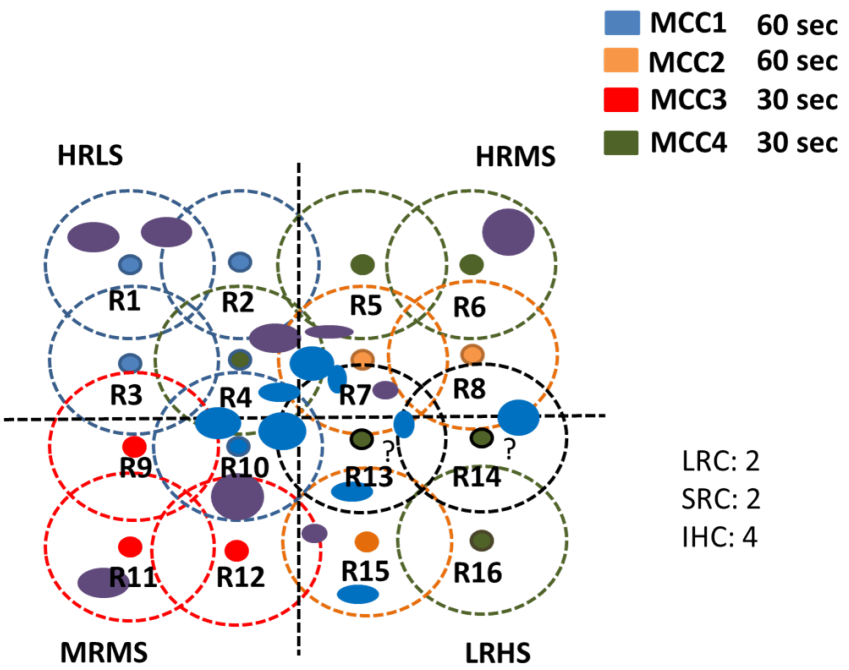


Figure A.6: MCC-Radar configuration for the heterogeneous scenario at time 0 sec.

MCC_1: 'Move R_4 from MCC_1 to MCC_4'

MCC_2: 'Move R_5 and R_6 from MCC_2 to MCC_4'

MCC_3: $\emptyset$ (It chooses to 'Use 60 second heartbeat')

MCC_4: 'Move R_{13} and R_{16} from MCC_4 to MCC_1' 'Move R_{14} and R_{15} from MCC_4 to MCC_2'

The numbers of all the types of conflicts are:

LRC: 2 (LRC_1: MCC_1 and MCC_2 move radars simultaneously to MCC_4; LRC_2: MCC_3 and MCC_4 move radars simultaneously to MCC_1.)

SRC: 0

IHC: 3 (IHC_1: MCC_1 and MCC_4 use different heartbeats; IHC_2: MCC_2 and MCC_4 use different heartbeats; IHC_3: MCC_3 and MCC_4 use different heartbeats.)

$PCR(\xi) = \frac{4*2+2*1+1*0}{4*2+2*4+1*2} = 0.56$, $\rho(0.44) = (-0.2) * 0.44 + 0.8 = 0.712$. Since $PCR(\xi) < \rho(t)$, each MCC unrolls its special state that includes overlapping context among neighbors (line 17-19, Algorithm 3), applies appropriate heuristic to select

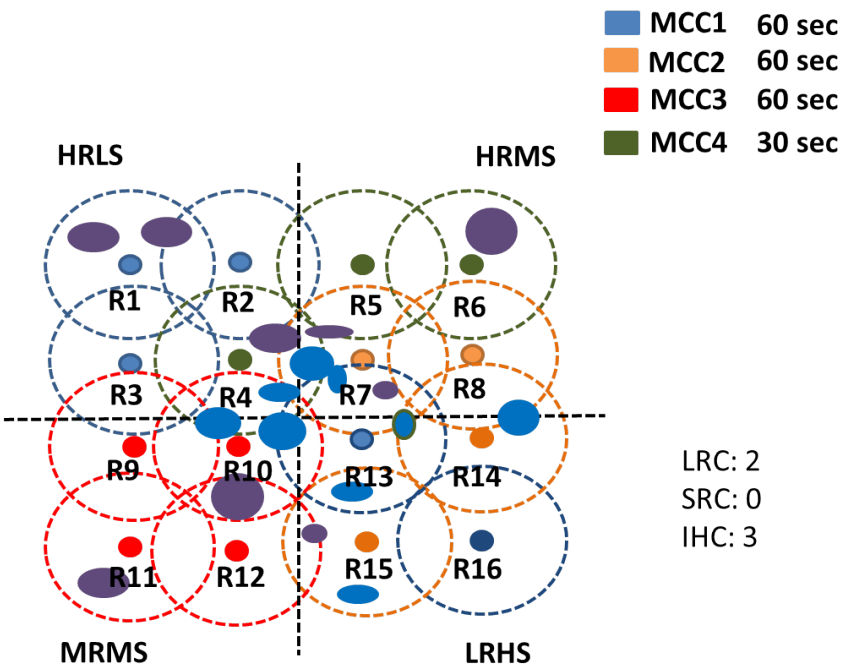


Figure A.7: MCC-Radar configuration for the heterogeneous scenario at time 0.44 sec.

another action for conflict resolution (line 20, Algorithm 3). apply-heuristic() is the function for MCC_i that chooses the heuristic with the highest priority among its set of heuristics. For more details about heuristics, see Chapter 5.3.5.

Time 0.52 sec (MMLC phase):

MCC_1's state S_9 (Fig A.8):

$F_0 : (HRLS, 60seconds, many)$;

$F_1 : \langle(60seconds, 1), (60seconds, 1), (60seconds, many)\rangle$;

$F_2 : \langle Low, Low, High\rangle$;

$F_3 : \langle HRMS, MRMS, LRHS\rangle$. (The weather scenarios of MCC_2, MCC_3 and MCC_4 are HRMS, MRMS and LRHS respectively.)

MCC_2's state:

$F_0 : (HRMS, 60seconds, 1)$;

$F_1 : \langle(60seconds, many), (60seconds, 1), (60seconds, many)\rangle$;

$F_2 : \langle Low, Low, Medium\rangle$;

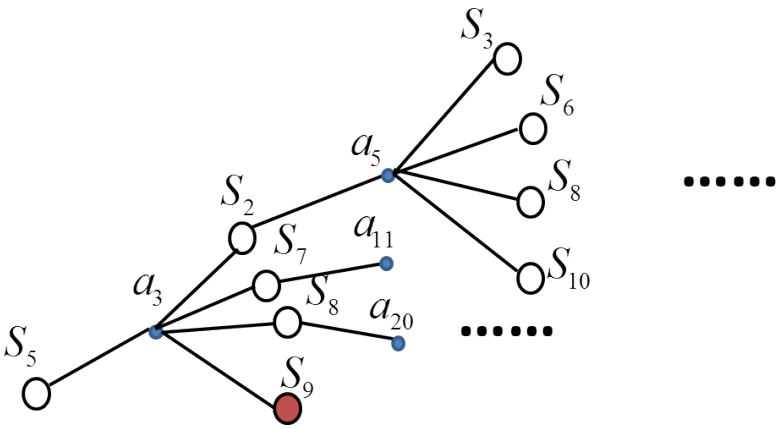


Figure A.8: The MDP space for MCC_1 at time 0.52 sec.

F_3 : $\langle HRLS, MRMS, LRHS\rangle$. (The weather scenarios of MCC_1, MCC_3 and MCC_4 are HRLS, MRMS and LRHS respectively.)

MCC_3's state:

F_0 : $(MRMS, 60seconds, 1)$;

F_1 : $\langle(60seconds, many), (60seconds, 1), (60seconds, many)\rangle$;

F_2 : $\langle Low, Low, Medium\rangle$;

F_3 : $\langle HRLS, HRMS, LRHS\rangle$. (The weather scenarios of MCC_1, MCC_2 and MCC_4 are HRLS, HRMS and LRHS respectively.)

MCC_4's state:

F_0 : $(LRHS, 60seconds, many)$;

F_1 : $\langle(60seconds, many), (60seconds, 1), (60seconds, 1)\rangle$;

F_2 : $\langle High, Medium, Medium\rangle$;

F_3 : $\langle HRLS, HRMS, MRMS\rangle$. (The weather scenarios of MCC_1, MCC_2 and MCC_3 are HRLS, HRMS and MRMS respectively.)

MCC_1:

Priorities of heuristics: $\{H_1 : 0.84;\ H_2 : 0;\ H_3 : 0.21;\ H_4 : 0.55;\ H_5 : 0.1;\ H_6 : 0.73;\ H_7 : 0.28;\ H_8 : 0.6\}$

Heuristic chosen: H_1; (It is defined in Chapter 5.3.5)

Previous action a_1: 'Light Move (MCC_1 to MCC_4)''Use 60 seconds heartbeat';

Updated action a_7: 'Heavy Move (MCC_1 to MCC_4)''Use 60 seconds heartbeat'.

MCC_2:

Priorities of heuristics: $\{H_1 : 0.28;\ H_2 : 0.39;\ H_3 : 0.93;\ H_4 : 0.48;\ H_5 : 0;\ H_6 : 0.12;\ H_7 : 0;\ H_8 : 0.3\}$

Heuristic chosen: H_3; (It is defined in Chapter 5.3.5)

Previous action : 'Heavy Move (MCC_2 to MCC_4)' 'Use 60 seconds heartbeat';

Updated action: 'Heavy Move (MCC_2 to MCC_4)' 'Use 30 seconds heartbeat'.

MCC_3:

Priorities of heuristics: $\{H_1 : 0.66;\ H_2 : 0.49;\ H_3 : 0;\ H_4 : 0.8;\ H_5 : 0.16;\ H_6 : 0.5;\ H_7 : 0.72;\ H_8 : 0.31\}$

Heuristic chosen: H_4; (H_4 is the heuristic that the agent unrolls its MDP space by exploring a new action that has a different type as well as a different direction of radar moves from the current action choice)

Previous action : 'Light Move (MCC_3 to MCC_1)' 'Use 30 seconds heartbeat';

Updated action : 'Heavy Move (MCC_3 to MCC_2)' 'Use 30 seconds heartbeat'.

MCC_4:

Priorities of heuristics: $\{H_1 : 0;\ H_2 : 0.29;\ H_3 : 0.54;\ H_4 : 0.62;\ H_5 : 0.1;\ H_6 : 0;\ H_7 : 0;\ H_8 : 0.75\}$

Heuristic chosen: H_8;

Previous action : 'Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat';

Updated action: 'Heavy Move (MCC_4 to MCC_2)' 'Use 30 seconds heartbeat'. (The heuristic H_8 is defined as: 'No change of action')

For MCC_1, S_9 is a state that has not been encountered during offline learning. S_9 is unrolled in the S^{init} of MCC_1 (Fig A.8). Similarly, MCC_2, MCC_3 and MCC_4 all unroll their new states in their initial MDP space. Since there are conflicts among the actions of MCCs, each MCC executes the iterations of UMDP and CR stages (line

22, Algorithm 3). Each MCC executes the decentralized negotiation algorithm (Procedure 4) to resolve conflicts from a partially global perspective. During the decentralized negotiation algorithm, MCCs update their detailed actions using a mediator-based mechanism so as to maximize the conflict resolution performance. The conflict resolution performance is measured by comparing two parameters: $PCR(\xi)$ (Chapter 5.3.1) and $\rho(t)$ (Chapter 5.3.1). $PCR(\xi)$ measures the conflict resolution performance in the neighborhood ξ by taking into account both the priorities and the number of the three types of conflicts. $\rho(t)$ is a variable that determines the threshold of conflict resolution performance. In this example, I initialize $PCR(\xi) = 1$ and $\rho(t) = -0.2 \times t + 0.8$. If $PCR(\xi) > \rho(t)$, the S^{init} of each MCC is unrolled using the method partial-unroll$(s', a, model)$ (line 3-4, Procedure 3). partial-unroll$(s', a, model)$ is a function that unrolls the action a and its subsequent search tree. For each unrolled state s_{succ}: if s_{succ} is an internal state and explored by offline learning (line 7-8, Procedure 1), it chooses the dominant action (the action with the highest probability distribution in its offline policy $\pi(s_{succ})$) to unroll (line 3, Procedure 1). The process continues until all the terminal states are reached. partial-unroll$(s', a, model)$ is a greedy method that always unrolls the branch of search space that has good performance during offline learning. If $PCR(\xi) \leq \rho(t)$, the MCC selects another action for conflict resolution by applying appropriate heuristics (line 7-8, Procedure 3). A set of heuristics with different priorities are used to guide the unrolling for each MCC. For example, H_1 is the heuristic that the agent unrolls its search space by exploring a new action that has a different type of radar moves from the current action choice. The priorities are updated based on the conflict resolution performance (line 5, Procedure 3). Procedure 5 computes the priority of each heuristic. Each MCC applies the heuristic that has the highest priority (line 7, Procedure 3). It ensures that the actions that have better performance on conflict resolution are more likely to be unrolled. This is an advantage over complete unrolling since it reduces the MDP

space significantly.

Since the *termination condition* is not met , each MCC enters the first round of UMDP and CR. After that, the detailed actions of all MCCs are updated as:

Time 1.1 sec (MMLC phase):

Round 1 (UMDP and CR):

MCC_1: 'Move R_1 from MCC_1 to MCC_4'

MCC_2: 'Move R_5 and R_7 from MCC_2 to MCC_4'

MCC_3: 'Move R_{11} from MCC_3 to MCC_2' (It chooses to 'Use 60 second heartbeat')

MCC_4: 'Move R_{13} and R_{14} from MCC_4 to MCC_2'

The numbers of all the types of conflicts are:

LRC: 0

SRC: 0

IHC: 3 (IHC_1: MCC_1 and MCC_2 use different heartbeats; IHC_2: MCC_1 and MCC_3 use different heartbeats; IHC_3: MCC_1 and MCC_4 use different heartbeats.)

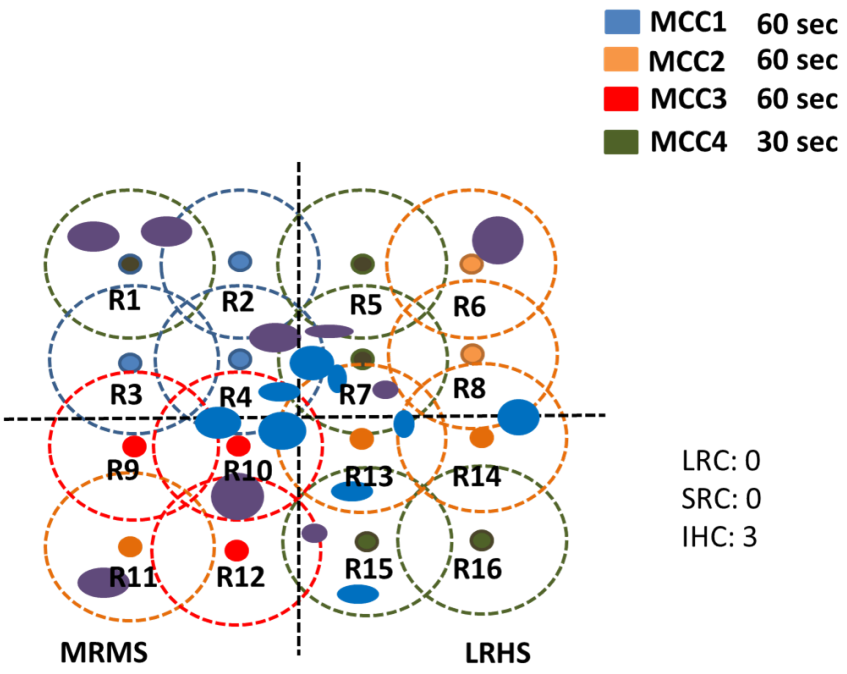


Figure A.9: MCC-Radar configuration for the heterogeneous scenario at time 1.1 sec.

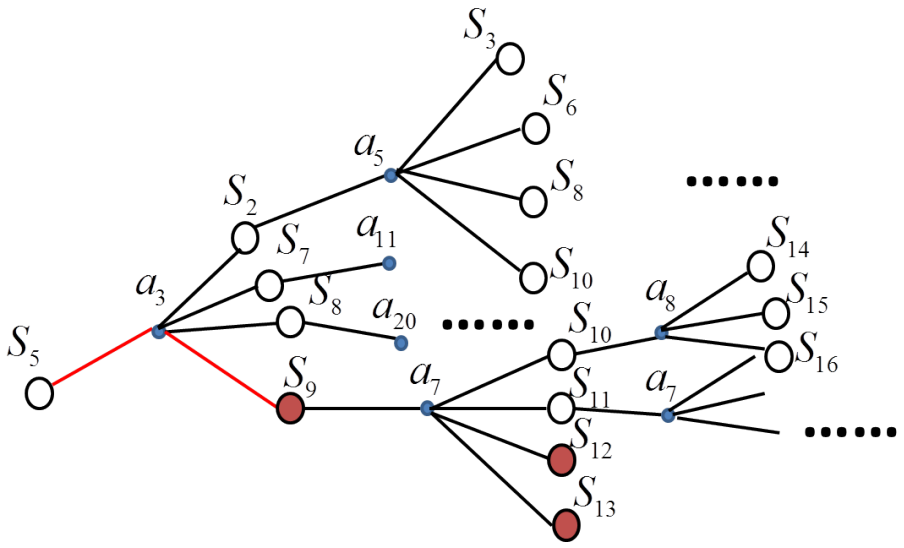


Figure A.10: The MDP space for MCC_1 at time t_{k+2}. The special states are marked red.

$PCR(\xi) = \frac{4*2+2*1+1*2}{4*2+2*4+1*2} = 0.67$, $\rho(1.1) = (-0.2) * 1.1 + 0.8 = 0.58$. Since $PCR(\xi) > \rho(t)$, each MCC unrolls its MDP space and updates the priorities of heuristics (line 4-5, Procedure 3). Figure A.10 shows the MDP space for MCC_1 after the first round of UMDP and CR. In Figure A.10, S_{12} and S_{13} are the special states that are not encountered during offline learning. The unrolling stops at the edge of S_{12} and S_{13}. When MCC_1 reaches these states during online learning, it will unroll subsequent search space based on the conflict resolution performance.

The updated priorities of heuristics for each MCC are:

MCC_1:

Priorities of heuristics: $\{H_1 : 0.88;\ H_2 : 0;\ H_3 : 0.21;\ H_4 : 0.55;\ H_5 : 0.1;\ H_6 : 0.73;\ H_7 : 0.28;\ H_8 : 0.6\}$

MCC_2:

Priorities of heuristics: $\{H_1 : 0.28;\ H_2 : 0.39;\ H_3 : 0.94;\ H_4 : 0.48;\ H_5 : 0;\ H_6 : 0.12;\ H_7 : 0;\ H_8 : 0.3\}$

MCC_3:

Priorities of heuristics: $\{H_1 : 0.66;\ H_2 : 0.49;\ H_3 : 0;\ H_4 : 0.82;\ H_5 : 0.16;\ H_6 : 0.5;$

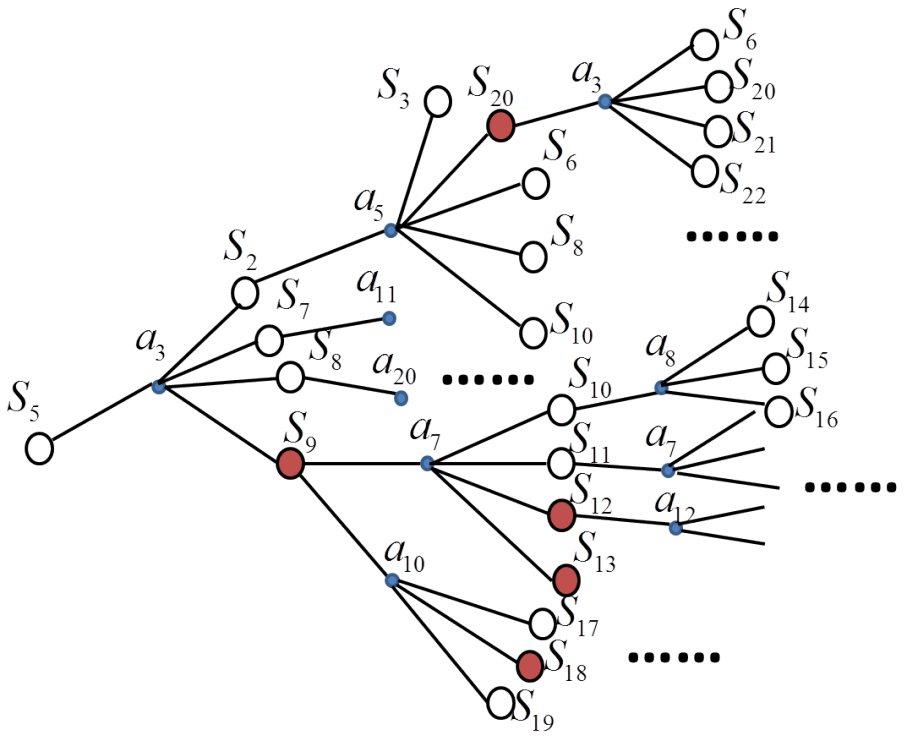


Figure A.11: The MDP space for MCC_1 after a few learning episodes.

$H_7 : 0.72$; $H_8 : 0.31\}$

MCC_4:

Priorities of heuristics: $\{H_1 : 0$; $H_2 : 0.29$; $H_3 : 0.54$; $H_4 : 0.62$; $H_5 : 0.1$; $H_6 : 0$; $H_7 : 0$; $H_8 : 0.76\}$

Since the *termination condition* is met, the iterations of UMDP and CR terminate after the first round. Fig A.11 shows the MDP space for MCC_1 after a few learning episodes. I can see that when action a_7 fails to resolve conflicts efficiently, the state S_9 applies heuristic to choose another action a_{10} and unrolls the subsequent search space. The updated heuristics help to unroll the most prominent actions and their subsequent search space and learn online policies for the new part of the MDP tree. S_{18} and S_{20} are new special states that are added to the problem.