FACULTY OF ENGINEERING
CAIRO UNIVERSITY

REINFORCEMENT LEARNING
ALGORITHMS FOR MULTI-ROBOT
ORGANIZATION

by

MOHAMMED EL-SAID I. EL-TELBANY

A Thesis Submitted to
Faculty of Engineering of Cairo University
In fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY
In
ELECTRONICS AND COMMUNICATIONS ENGINEERING

UNDER THE SUPERVISION OF

Prof. Dr. Samir Ibrahim Shaheen
Computers Engineering Dept.
Faculty of Engineering
Cairo University
Giza, Egypt

Prof. Dr. Ashraf H. Abdelwahab
Computers and Systems Dept.
Electronics Research Institute
Giza, Egypt

Faculty of Engineering, Cairo University,
Giza, Egypt.
June 2003
The decentralized control and distributed artificial intelligence became common in everyday systems, where there are techniques for coordinating multiple intelligent mobile robots gaining popularity in the robotic community. Certainly, a decentralized and distributed solution is the only viable approach for many real-world application domains that are inherently distributed in time, space, and/or behaviorally. Therefore, there has been an upsurge of interest in multiple autonomous mobile robots engaged in collective behavior that supporting and complementing each other. Many of these real-world applications are space missions, operations in hazardous environments (fire fighting, cleanup of toxic waste, nuclear power plant decommissioning, security), and military operations. The idea that group(s) of cooperating intelligent robots can be more flexible, more reliable and more fault-tolerant, and more economical than a single, monolithic robot only and only if efficient means can be found for cooperating robot group(s). However, one of several problems exist in multiple autonomous mobile robots engaged in cooperative behavior is dynamic organization, which is the difficulty in determining the proper coordination and task-allocation schemes during task execution in response to conditions in the environment and within the team itself. Solving this problem is a big challenge, and will lead to an optimal cooperative robot team(s). Consequently, if solved, it will assist in the achievement of group(s) goal rapidly, optimize the global system behavior by reducing interference and conflicts that arise between robots, deadlock or stagnation, and oscillations. Certain approaches have proven quite successful and have caused a re-analysis within the field of artificial intelligence of what components are necessary in the intellectual architecture of such multi-robot systems. The goal is to build robots that come close to what the people wish. Researches suggested that the learning incitement multi-robot to behave efficiently in cooperative and organizational manner to optimize the system behavior. One of these proposed learning techniques is reinforcement learning algorithms which have received increased attention as a method for robot learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors. In this case, the reinforcement signal directly evaluates the robots behaviors. Using reinforcement learning can acquire an organization behavior experience from multi-agent robotics environments and attempts to direct the robots behavior away from individual greediness towards global efficiency, and reducing the robots’ interference, conflicts, and increasing cooperative and adaptive character automatically. This will lead to a more efficient coherence as demonstrated by high efficiency
and decrease in execution time. The research proposed in this thesis presents new distributed reinforcement learning algorithms that acquires cooperative and organizational behavior experience in multi-agent robotics environments. The major objective of this work is to provide a framework that makes the use of reinforcement learning techniques on multi-robots effective. These algorithms help to reduce the size of the search area, and hence the search time, by providing the learner(s) with auxiliary sources of bias (i.e., rewards and experiences). The experimental results demonstrate superior performance for distributed reward and experiences, strong performance for distributed experiences and good performance for distributed rewards. Moreover, a new proposed algorithm for dynamic task allocation between multiple robots is implemented and tested on foraging and box-pushing task domain to study the effect of coordination and heterogeneity on the allocation performance. The contributions can be summarized as follows:

I. A novel formulation for distributed reinforcement learning, by introducing the learning from distributed rewards. Novel algorithms to adaptively estimates the weights of distribution using simulated annealing are also presented.

II. A novel algorithm to integrate the apprentice and reinforcement learning to speed up the learning activity.

III. A novel algorithm integrates the learning from both distributed rewards and experiences to get the merits of both techniques.

IV. A framework for integrating market-based reinforcement learning for task-allocation among multi-robot systems.


ACKNOWLEDGEMENT

My parents over the years shown an amount of patience that is hard for me to fathom. They consistently gave me the freedom to chart my own path and stood by me, and without their support none of this would have been possible.

There are many people to thank for contributing to this dissertation and to my success in graduate study. Foremost is professor Samir I. Shaheen, who has been my principle guide throughout my graduate career. I have benefited enormously from his scientific curiosity, attention to details, and research standard. He has considerably changed my view to the world. Professor Ashraf H. Abdelwahab has provided invaluable technical assistance over the years, improving my writing style, and gives me the moral support needed to complete this dissertation. Finally, I cannot adequately express my gratitude to Maha A. Maged, who has tolerated me throughout the work that led to this dissertation.
# TABLE OF CONTENTS

1. **Introduction**
   1.1 Cooperation and Organization ................................................................. 2
   1.2 Communication Techniques in Robotics ....................................................... 2
   1.3 Machine Learning Techniques ...................................................................... 3
   1.4 Research Objective ....................................................................................... 5
   1.5 Contributions .............................................................................................. 6
   1.6 Thesis Organization ..................................................................................... 7

2. **Behavior-Based Control**
   2.1 Behavior-Based Robotics ............................................................................. 10
   2.2 Subsumption Control Architecture ............................................................. 12
   2.3 Potential Fields .......................................................................................... 13
   2.4 Schema-Based Reactive Control .................................................................. 15
   2.5 Temporal Sequencing ................................................................................ 17
   2.6 Motor Schema Versus Subsumption Architecture ........................................ 18
   2.7 Conclusion ................................................................................................ 19

3. **Reinforcement Learning and Cooperation**
   3.1 Introduction ................................................................................................ 21
   3.2 Reinforcement Learning .............................................................................. 22
   3.3 Monte Carlo Learning ............................................................................... 26
   3.4 Temporal Difference Learning .................................................................. 27
   3.5 Q-learning Algorithm .............................................................................. 28
   3.6 SARSA Learning Algorithm .................................................................... 30
   3.7 Which Algorithm Should We Use? ............................................................. 31
   3.8 Planning in Reinforcement Learning ......................................................... 31
   3.9 Designing the Reinforcement Functions .................................................... 32
   3.10 Other Reinforcement Learning Techniques ............................................. 33
       3.10.1 Evolutionary Reinforcement Learning ............................................. 34
       3.10.2 Statistical Reinforcement Learning ............................................... 35
   3.11 Comparison Between Learning Techniques ............................................. 35
   3.12 Cooperation in Multi-Robot Systems .................................................... 37
   3.13 Reinforcement Learning in Multi-Robot Systems ..................................... 39
   3.14 Task-Allocation and Cooperation ......................................................... 45
   3.15 Discussion and Summary ....................................................................... 47

4. **Learning Cooperation by Rewards Distribution**
   4.1 Introduction .............................................................................................. 51
   4.2 Distributed Q-Learning ............................................................................ 53
   4.3 Distributed Rewards Weighting Functions ............................................... 57
   4.4 Learning Adaptive Distribution of Rewards .............................................. 59
   4.5 Experiments Result ............................................................................... 61
4.5.1 Domain of Experiments ............................................................... 61
4.5.2 Performance Metrics of Learning Strategies ........................................ 62
4.5.3 Experiments Setup ................................................................................. 62
4.6 Experiments ................................................................................................. 63
4.6.1 Case I: Human Tuning for Weighting Parameters .................................. 63
4.6.2 Case II: Adaptive Tuning for Weighting Parameters......................... 65
4.6.3 Case III: The Effect of Communication Range on Performance .......... 69
4.7 Discussion and Conclusions ....................................................................... 71

5. Integrating Apprentice and Reinforcement Learning in Multi-Robot Domains
   73
5.1 Introduction ............................................................................................................. 73
5.2 Learning by Sharing Successful Expertise ................................................................. 76
5.3 Learning from Expertise and Rewards ................................................................. 77
5.4 Experiments Results ............................................................................................... 78
5.4.1 Case I: Learning from Training Experiences .............................................. 79
5.4.2 Case II: Learning from Training Experiences and Rewards .................... 82
5.5 Evaluating Reinforcement Learning Strategies ................................................... 83
5.6 Flexibility of Learning Strategies ......................................................................... 86
5.7 Multi-Robot Control Strategies: A Comparison ................................................. 88
5.7.1 Centralized Control ......................................................................................... 88
5.7.2 Decentralized Control .................................................................................. 90
5.8 Behavioral Difference ......................................................................................... 92
5.9 Discussion and Conclusions ................................................................................. 94

6. Market-Based Reinforcement Learning for Multi-Robot Task-Allocation
   97
6.1 Introduction ............................................................................................................. 97
6.2 Market-Based Reinforcement Learning for Task-Allocation ......................... 100
6.3 Experiments Validation ....................................................................................... 102
6.4 Experiments Design ............................................................................................. 103
6.4.1 Case I: Centralized Task-Allocation Among Homogenous Robots (CHM) .. 104
6.4.2 Case II: Distributed Task Allocation Among Homogenous Robots (DHM) .. 106
6.4.3 Case III: Centralized Task-Allocation Among Heterogeneous Robots (CHT) 108
6.4.4 Case IV: Distributed Task-Allocation Among Heterogeneous Robots (DHT) 108
6.5 Experiments Results ......................................................................................... 108
6.6 Discussion and Conclusions ................................................................................. 113

7. Conclusions
   115
7.1 Summarizing the Results ................................................................................... 115
7.2 Issues for Future Research ............................................................................... 118

A. Introduction to Behaviors Design and Programming
   121
A.1 Behavior Encoding ........................................................................................... 121
A.2 Concurrent Behaviors Combination ............................................................... 122
A.3 Subsumption Architecture Behaviors’ Programming ................................................. 123
A.4 Behaviors and Schema Theory ................................................................................ 125
A.5 Motor-Schema Behavior’ Programming ................................................................. 126
A.6 Motor-Schema Concurrent Behaviors Combination ............................................. 128

B. Multi-Robot Simulator Software .................................................................................. 131
   B.1 Teambots Simulation Environment ......................................................................... 131
   B.2 Foraging Task Behavioral Design ........................................................................ 133

C. Bounds with a Learning Rate $k$ .............................................................................. 135

References ..................................................................................................................... 137
LIST OF FIGURES

Figure 1.1: Dynamic team organization behavior gained through learning, which lets the robots complement each other in cooperative manner.........................................................3

Figure 2.1: Horizontal control architecture of the robot..........................................................................................................................11
Figure 2.2: Vertical control architecture of the robot. ..............................................................................................................................12
Figure 2.3: Behavior module as augmented FSA.................................................................12
Figure 2.4: (a) The configuration of obstacles (black) in the map. (b) The attractive potential associated with the goal configuration. (c) The repulsive potential associative with obstacles. (d) The sum of the attractive and repulsive potentials (adopted from Pirjanian, 1999).............................................................................................................................14
Figure 2.5: Motor schema example. The diagram on the left shows a vector corresponding to move-to-goal schema; the center diagram shows avoid-obstacles schema. On the right, the two schemas are summed, resulting in a complete behavior for reaching the goal.................................................................................................................................................16
Figure 2.6: The Forage finite state automaton (FSA)..............................................................................................................................17

Figure 3.1: The basic reinforcement-learning model. The robot interacts with an environment in a feedback cycle. .....................................................................................................................23
Figure 3.2: The Markov decision process model. .................................................................................................................................24
Figure 3.3: The L-ALLIANCE architecture (adopted from Parker, 1994)....................................44

Figure 4.1: Cooperative and Distributed $Q$-Learning........................................................................................................................54
Figure 4.2: Average number of collected attractors elements for various numbers of robots with different reward weighing value. ..........................................................................................63
Figure 4.3: The average number of attractors collected by robots teams in foraging task under local, global and distributed rewards tuned by human. .....................................................................................64
Figure 4.4: Convergence of learning for four robots under local, global and manually distributing rewards; lower numbers indicate convergence to a stable team policy........65
Figure 4.5: The time-to-convergence with four robots. Locally, globally and distributed reward stabilized but with different levels. ........................................................................................................66
Figure 4.6: The average number of attractors collected by robot teams in foraging task under manual weight equals three and automatically (static) adaptive weight for distributed rewards........................................................................................................................................67
Figure 4.7: Convergence of learning for under manually weighted of distributed rewards and static weight adaptation for distributed rewards algorithm; lower numbers indicate convergence to a stable team policy.................................................................68
Figure 4.8: The average number of attractors collected by robot teams in foraging task under static weight adaptation and dynamic weight adaptation for distributed rewards........68
Figure 4.9: Convergence of learning for under static weighting of distributed rewards and dynamic weighting for distributed rewards algorithm; lower numbers indicate convergence to a stable team policy.................................................................69
Figure 4.10: The number of trials needed for steady state throughput for different learning schemes........................................................................................................................................70
Figure 4.11: The effect of communication range on the distributed rewards reinforcement algorithm performance. .............................................................................................................70

Figure 5.1: The relationship between robots, the robots play two roles as a trainer and learner. ............................................................................................................................................74

Figure 5.2: The general architecture for the integrated learning and planning in multi-robot domains. ................................................................................................................................................77

Figure 5.3: The average number of attractors collected by robots in foraging task under different learning strategy. The training experiences enhanced the throughput of the team in comparison with local reward and distributed rewards policies. ........................................80

Figure 5.4: Convergence of learning for under automatic (dynamic) weight adaptation of distributed rewards and distributed experiences algorithm; lower numbers indicate convergence to a stable team policy. ................................................................................81

Figure 5.5: The time-to-converge rate curves for the four robots under learning from different strategy. ........................................................................................................................................81

Figure 5.6: The throughput of robots in foraging task under different learning strategy. ......................................................82

Figure 5.7: Convergence of learning for under distributed experiences and both of rewards and experiences distribution algorithms. ..........................................................................................................................83

Figure 5.8: The performance of different learning strategy in comparison with theoretically performance for the ideal cooperative teams. ..................................................................................84

Figure 5.9: The interference level under different learning strategy in comparison with theoretically ideal cooperative teams. ..........................................................................................................................85

Figure 5.10: The speedup metric of different learning strategy. .................................................................................................85

Figure 5.11: The performance of an added robot to a team of learned and non-learned robots using locally rewarded learning strategy. ..................................................................................................................86

Figure 5.12: The performance of an added robot to a team of learned and non-learned robots distributed rewards learning strategy. ..................................................................................................................87

Figure 5.13: The performance of an added robot to a team of learned and non-learned robots using distributed experiences learning strategy. ..................................................................................................................87

Figure 5.14: The performance of an added robot to a team of robots with different scenarios and learning strategy. .................................................................................................................................88

Figure 5.15: (a) The single centralized group policy, (b) The shared group policy. ..............................................................89

Figure 5.16: The average number of attractors collected by robot teams in foraging task using shared policy in comparison with other three learning strategies. .................................................................90

Figure 5.17: Convergence of learning for under shared policy in comparison with other three learning policies; lower numbers indicate convergence to a stable team policy. ........................................91

Figure 5.18: Decentralized Control Policies. .................................................................................................................................91

Figure 6.1: Learning dynamic task allocation based on can be viewed as an action selection problem where a robot is selecting a task to take so as to maximize its expected utility (Adopted from Gerkey and Matarić, 2002). .........................................................................................99

Figure 6.2: The simulation snapshots for (a) foraging task and (b) pushing-box with foraging. ................................................104

Figure 6.3: The control architecture and centralized market-based reinforcement learning behaviors. ......................................................................................................................................................106
Figure 6.4: The distributed control architecture permits the robots to engage in the same task at the same time. In the upper-left corner two robots respond to call-for-help and went to the failure robot simultaneously.................................................................107

Figure 6.5: The average response time needed by the team to converge to stable policy over time of robots’ team in the four cases.................................................................109

Figure 6.6: The success rate of robots team in the four cases; this is measured by the percentage allocated tasks to announced tasks.................................................................110

Figure 6.7: This sequence of snapshots shows the progress of leaning process of task-allocation in homogenous robot’s team. .................................................................111

Figure 6.8: This sequence of snapshots shows the progress of leaning process of task-allocation in homogenous robot’s team. .................................................................112

Figure A.1: Simple Behavior Diagram. .................................................................121

Figure A.2: The classification of multiple behaviors coordination techniques (adopted from Pirjanian, 1999). .................................................................123

Figure A.3: Subsumption based model for foraging robot (adopted from Arkin 1998). ..........124

Figure A.4: Behavior decomposed into perceptual and motor schema (adopted from Murphy, 2000). .................................................................125

Figure A.5: Five primitive potential fields (a) uniform, b) perpendicular, c) attractive, d) repulsive, and e) tangential (adopted from Murphy, 2000). .................................................................126

Figure A.6: Behavior Fusion via vector summation (adopted from Arkin, 1998). ..........128

Figure B.1: The Teambots Simulation Environment.................................................................131

Figure B.2: The Teambots real robot.................................................................132
LIST OF TABLES

Table 2.1: Comparison between Subsumption architecture and Motor schema. .................................18
Table 3.1: First-visit Monte Carlo method for estimating $V^z(s)$. ....................................................26
Table 3.2: TD (0) learning algorithm for estimating $V(s)$. ...............................................................28
Table 3.3: Q-learning algorithm. ........................................................................................................29
Table 3.4: Q-Dyna learning algorithm. ...............................................................................................32
Table 3.5: Comparison between different reinforcement techniques in a set of characteristics. 36
Table 3.6: The Statistics Parameters Meanings....................................................................................43
Table 4.1: A summary of the situated learning problem addressed here, and the structure of the proposed solution. ........................................................................................................52
Table 4.2: Summary of four learning schemes. ..................................................................................58
Table 4.3: Static linear estimation of the selfishness gain $w$ using simulated annealing. ...............60
Table 4.4: Dynamic estimation of the selfishness gain $w$ using simulated annealing. .................61
Table 4.5: The best estimated weights’ values for seven robots team..............................................66
Table 5.1: A summary of the situated learning problem addressed here, and the structure of the proposed solution. ........................................................................................................75
Table 5.2: A novel algorithm for integrated apprentice learning with reinforcement learning in multi-robot domains. ........................................................................................................78
Table 5.3: A novel algorithm for integrated social and distributed rewards learning in multi-robot domains. ........................................................................................................................79
Table 5.4: Comparison between different multi-robot control strategies. ....................................92
Table 5.5: The learned policies of two foraging robots. .....................................................................93
Table 6.1: A summary of the situated learning problem addressed here, and the structure of the proposed solution. ........................................................................................................100
Table 6.2: Description of the perceptual schemas designed and used in centralized task-allocation in foraging task..........................................................................................................105
Table 6.3: Description of the motor-schemas designed and used in centralized task-allocation in foraging task. .................................................................................................................105
Table 6.4: Description of the perceptual schemas designed and used in distributed task-allocation in foraging task. .......................................................................................................106
Table 6.5: Description of the motor-schemas designed and used in distributed task-allocation in foraging task. ...............................................................................................................107
Table 6.6: Description of the perceptual schemas used in centralized task-allocation in box- pushing task. ..................................................................................................................108
Table 6.7: Description of the motor-schemas used in centralized task-allocation in box-pushing task. ..........................................................................................................................108
CHAPTER 1

INTRODUCTION

The decentralized control and distributed artificial intelligence became common in everyday systems, where there are techniques for coordinating multiple intelligent mobile robots gaining popularity in the robotic community. Certainly, a decentralized and distributed solution is the only viable approach for many real-world application domains that are inherently distributed in time, space, and/or behaviorally. Therefore, there has been an upsurge of interest in multiple autonomous mobile robots engaged in collective behavior that supporting and complementing each other. Many of these real-world applications are space missions, operations in hazardous environments (fire fighting, cleanup of toxic waste, nuclear power plant decommissioning, security), and military operations [Arkin, 1998; Balch and Arkin; 1998]. The idea that group(s) of cooperating intelligent robots can be more flexible, more reliable and more fault-tolerant, and more economical than a single, monolithic robot only and only if efficient means can be found for cooperating robot group(s) [Cao et al, 1997; Balch and Arkin, 1998]. However, one of several problems exist in multiple autonomous mobile robots engaged in cooperative behavior is dynamic organization, which is the difficulty in determining the proper coordination and task-allocation schemes during task execution in response to conditions in the environment and within the team itself. Solving this problem is a big challenge, and will lead to an optimal cooperative robot team(s). Consequently, if solved, it will assist in the achievement of group(s) goal rapidly, optimize the global system behavior by reducing interference and conflicts that arise between robots, deadlock or stagnation, and oscillations [Kube and Zhang, 1993; Kube and Bonabeau, 1998; Matarić, 1992; 1994a; Matarić, 1997a; Goldberg and Matarić, 1997]. Certain approaches have proven quite successful and have caused a re-analysis within the field of artificial intelligence of what components are necessary in the intellectual architecture of such multi-robot systems. The goal is to build robots that come close to what the people wish. Researches suggested that the learning incitement multi-robot to behave efficiently in cooperative manner to optimize the system behavior [Stone and Veleso, 1997; Weiss, 1997; Wiering et al., 2000]. One of these proposed learning techniques is reinforcement learning algorithms which have received increased attention as a method for robot learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors. In this case, the reinforcement signal directly
evaluates the robots behaviors. Using reinforcement learning can acquire an organization behavior experience from multi-agent robotics environments and attempts to direct the robots behavior away from individual greediness towards global efficiency, and reducing the robots’ interference, conflicts, and increasing cooperative and adaptive character automatically. This will lead to a more efficient coherence as demonstrated by high efficiency and decrease in execution time.

1.1 Cooperation and Organization

Cooperation is a vital part of applying multiple robots to a mission in an efficient manner. It is a form of interaction that lets the individual robot capable of collaborating with other robots to accomplish that are beyond individual’s capabilities. Designing and implementing cooperative team of robots needs to decide how to resolve organization problems. The learning of choosing organization scheme is based on the improvement gained to complete the mission cooperatively within the efficiency constraints (i.e., execution time, interference, and robustness). Gaining dynamic organization behavior through learning lets the robots complement each other in cooperative manner. This will exist different organizations, which have different behaviors, and this behavior difference leads to difference in performance. Therefore, no one organization is best suited for every mission. Dynamic assigning each robot different roles through learning allows a group of robots to improve their performance by providing constraints for robots by accepting tasks that meet their roles in the organization based upon these robots’ skills and experience. As shown in Figure 1.1, increasing the level of communication and dynamic task allocation leads to a virtual centralized controller that direct the multi-robot systems for a coherent behavior. The cooperative and adaptive character of robots behavior increases the organization level, which demonstrated by high efficiency and decreasing execution time. In addition, preserve for the robots in the team with their autonomous.

1.2 Communication Techniques in Robotics

Communication is necessary for cooperation among multiple robots if and only if the robots use the information that they communicate to enhance their performance as individuals and/or as a group. There are three kinds of communication used in robotics [Cao at. el., 1997]. Explicit communication occurs when symbols representing data or commands are
exchanged between robots. This active communication may be local or global (broadcast). *Implicit communication* happens when robots gain useful information by observing each other performing actions in the environment and understanding of a robot’s actions. Using implicit communication, robots cooperate by predicting the behavior, *awareness* of other robots or estimating reinforcements of other from past observations and this type of information does not require the robot to model another robot’s internal state [Parker, 1995a; Matraić, 1994a; 1997a]. This form of communication is passive and local communication occurs when individuals sense other individuals.

![Figure 1.1: Dynamic team organization behavior gained through learning, which lets the robots complement each other in cooperative manner.](image)

*Stigmergic* communication is when a robot gains useful information by observing the changes other robot has made to the environment. Considering the environment as an implicit communication channel that provides awareness of robot by another; usually it refers to communication based on modification of the environment. All three types of communication occur in the natural world. For instance, the honeybee’s dance and human language are explicit. Implicit communication occurs when you approach a door at the same time as another person and they hold open the door to allow you to pass through first. Stigmergy is seen when one ant follows the pheromone trial another one laid to find food. Neither ant ever communicated directly with the other – they only changed or observed the change in the environment [Cao et al., 1997].

### 1.3 Machine Learning Techniques

Research in machine learning has undergone a tremendous growth. This field is contained within the broader confidences of artificial intelligence, and its attraction for researchers
tems from many resources. Machine learning focused on developing approaches and algorithms for artificial agent (i.e., robot) to automatically acquiring knowledge, developing strategies and modifying its behavioral tendency by experience. This is especially important in multi-robot systems, where learning has often been neglected. The designer of multi-robot systems have generally taken on the extremely difficult task of trying to anticipate all possible contingencies and interactions among the robots ahead of time. Therefore, the machine learning ultimate goal is to self-organize or coordinate multiple intelligent robots to deal with noise in their internal and external sensors and with inconsistencies in the behavior of the environment and other robots, especially in domains have been characterized by friendly and hostile robots, inter-robot cooperation, real-time interaction, and dynamic, uncertain environments [Stone, 1998]. Machine learning has studied many different types of learning. They are classified under two types of learning: supervised and unsupervised [Russell and Norvig, 1995; Russell, 1996; Mahadevan, 1996a; Dietterich, 1997; Mitchell, 1997].

- **In supervised learning**, a designer carefully selects training examples of the task to be performed by robots, which must be supplied by an expert human or computerized. The backpropagation algorithm for training artificial neural networks is one common technique for supervised learning [Mitchell, 1997; Haykin, 1999]. If the training information provided is more limited in the form of scalar, performance-related reward, not indicate the correct action itself then learning method become a reinforcement learning. ALVINN (Autonomous Land Vehicle In a Neural Network) is a well-known application of supervised learning to robot control [Pomerleau, 1993]. The network receives as input an image of the road ahead and produces, as output the steering command required keeping the car on the road.

- **Reinforcement learning** algorithms can be used as a component of multi-robot systems. Where the environment supplies the scalar reward signal indicates how good or bad the robot's action was. The mapping from state to action performed by the robot, the policy, is the object modified during learning [Nehmzov and Mitchell, 1995]. Usually the policy is based on a value function providing the estimate of the expected future reward to receive in a state or for a state-action pair. The reward is said to be immediate when, and only when an optimal policy is always taking action with the highest next reward without considering future consequences. If such a policy is not optimal, rewards are delayed. The reinforcement learning problems can be solved in principle using dynamic programming algorithms [Barto at. el., 1995; Sutton and Barto 1998]. However, dynamic
programming requires a model of the environment, which is not available in dynamic
interactive tasks and time that is polynomial in the number of the states. Some recent
reinforcement learning algorithms have been designed to perform dynamic
programming in an incremental manner without the need of state transition
probabilities and reward structure of the environment by enhancing the performance
on-line while interacting with the environment. Reinforcement learning algorithms can
be used as components of multi-robot algorithms. If the members of a group of robots
each employ one of these algorithms, a new collective algorithm emerges for the group as
a whole. This type of collective algorithm allows control policies to be learned in a
decentralized way.

- **Unsupervised learning**, on the other hand, doesn’t provide a feedback on the learning
task. Unsupervised learning techniques perform a clustering of incoming information
without knowing the target output. Unsupervised learning can identify clusters of
similar data points; enable the data to be represented in terms of more orthogonal
features. This reduces the effective dimensionality of the data, enabling representation
that is more concise and supporting more accurate supervised learning [Mitchell, 1997;
Haykin, 1999]. One of unsupervised learning paradigm is Kohonen map, which is
frequently used, in autonomous robots, most often for categorization, map-building
and motor control tasks. A well-known application of Kohonen network for location
recognition. The robot learns to recognize a particular location based on information
from motor signals [Nehmzow and Smithers, 1992].

### 1.4 Research Objective

**Problem Definition**: Given a specific mission, how can the relationship between robots organization and
reinforcement learning in multi-robot systems be investigated in principled manner?

The major objective of this work is to provide a framework that makes the use of
reinforcement learning techniques on multi-robots effective. Throughout this dissertation, a
particular view of learning among multiple robots is taken. We assume that our goal is to
produce good control policies for dynamic organization of multiple autonomous robots that
need to coordinate their behaviors in cooperative manner in pursuit of a common goal(s).
Different reinforcement-learning algorithms are proposed, implemented and analyzed. The
proposed algorithms based on distributing the centralized optimal policy to near optimal
decentralized optimal policies. This done by rewards and experiences distribution comes from other team members. The proposed algorithms were tested on foraging task domain to achieve a broader understanding of how reinforcement learning organizes individual robots to produce coherent cooperation and the reinforcement-learning algorithms incitement multi-robot teams to behave efficiently in a cooperative manner, especially when the characteristics of the robot society are unknown, or because they change over time. Moreover, a new proposed algorithm for dynamic task allocation is implemented and tested on foraging and box-pushing task domain to study the effect of coordination and heterogeneity on the allocation performance. The research addresses these points by applying a principled approach to the analysis and design of reinforcement learning algorithms to acquire organization behavior in multi-robot teams.

1.5 Contributions

The field of reinforcement learning is currently facing the challenge of moving from theory to real-world applications. This dissertation explores the performance of reinforcement learning algorithms used for cooperative multi-robot systems. The ultimate goal is to formulate a new reinforcement learning algorithm(s) that helps in solving the dynamic organization problem, and to exploring the effect of reinforcement functions in reaching the above goal. The work involves several extensions over the usual reinforcement-learning framework, including the distribution reinforcement learning, and task-allocation and, so on. Contributions of this are divided into two categories: first, soft contributions: the viewpoints, taxonomies and comparisons; and second, hard contributions: the algorithms and experimental results. The soft contributions of this research are:

I. Identify the characteristics of Subsumption architecture and Motor Schema control in a fair comparison (cf. §2.6).

II. Provide taxonomy for reinforcement learning techniques with identifying their pros and cons (cf. §3.10 and §3.11).

III. Identify the different types of extra-work needed for incitement the cooperation among the multi-robots (cf. §4.1).

The hard or technical contributions of this dissertation are summarized as follow:
V. A novel formulation for distributed reinforcement learning, by introducing the learning from distributed rewards. Novel algorithms to adaptively estimates the weights of distribution using simulated annealing are also presented (cf. chapter 4).

VI. A novel algorithm to integrate the apprentice and reinforcement learning to speed up the learning activity (cf. chapter 5).

VII. A novel algorithm integrates the learning from both distributed rewards and experiences to get the merits of both techniques (cf. chapter 5).

VIII. A framework for integrating market-based reinforcement learning for task-allocation among multi-robot systems (cf. chapter 6).

1.6 Thesis Organization

The thesis is organized as follows:
Chapter 2 describes the behavior-based control in comparison with tradition control. Moreover two of the main control paradigms described to differentiate between them using their characteristics are presented and discussed.

Chapter 3 describes the Markov based reinforcement learning algorithms and other reinforcement techniques with a comparison among them to highlighting their pros and cons. Moreover, why the Q-learning is used as our reinforcement technique with issues in designing reward functions, especially in multi-robot domain. Background and related works in the field of behavior-based robotic is also presented including: behavior-based multi-robot systems cooperation and learning, describing the limitations of the system developed so far, and point toward dynamic organization as a solution to some of their problems.

Chapter 4 introduces the proposed novel algorithms for learning cooperation behavior among multi-agent robots by distributing their rewards. In addition, a new methodology to estimate the adaptive weights of these rewards is presented. This chapter also verifies these algorithms by empirical results for foraging domains.

Chapter 5 introduces the proposed novel algorithms for learning from training experiences of other team members as an apprentice learning. Moreover, a new algorithm integrates both of learning from distributed rewards and training experiences is presented. These algorithms are implemented and tested using foraging domains. Empirical results are showed their success as compared to proposed learning algorithms in previous chapter.

Chapter 6 introduces a new methodology for dynamic task-allocation among multiple robots based on market based reinforcement learning. The chapter verifies this methodology by empirical
results for foraging and box-pushing domains to show its effectiveness on heterogeneity and coordination levels.

Chapter 7 summarizes the work performed, the major contributions of this thesis, and outlines the most promising directions for future research.

Finally, Appendices A, B and C are added to demonstrate the programming requirements for the behavior-based control, describe the simulator software used in this work, and the proof of convergence for the market-based reinforcement learning algorithm.
CHAPTER 2

BEHAVIOR-BASED CONTROL

Building a group of robots that shows a cooperative behavior while executing a complex task is an interesting research challenge. In the same time, there has been an upsurge of interest in learning in autonomous mobile robots deals with the behavior-based paradigm. This approach concentrates on physical systems situated in the real world and promotes simple associative learning between sensing and acting. The behavior-based control has been served as an effective methodology for multiple robot control and learning in a large number of multi-robot problems domains. This chapter presents and overviews the behavior-based control methods and a comparison between them to show their pros and cons. The goal is to establish a base from which our research proceeds.

2.1 Behavior-Based Robotics

There are two control approaches used for robot control, traditional approach and behavior-based approach. The traditional approach [Akin, 1998; Russell and Norvig, 1995] structuring a robot’s controls into functional modules – perception, planning, learning etc., and constraining as much as possible the environment where the robot will operate. Creating a model of the environment and the robot preprocess sensor information into abstracted internal representation that are acted on by a central planner, then instantiated the results to become actions that can be executed by the robot to reach a specific goal. This can be represented in horizontal control architecture as shown in Figure 2.1. Control systems using this architecture solve their task in several steps. First, the sensor input is used o modify the internal representation of the environment. Second, based on the internal representation planning is made. This results in a series of actions for the robot to take to reach a specified goal. Third, this series of actions is used to control the motors of the robot. This completes the cycle of the control system and it’s restarted to achieve new goals. This approach has several problems. Maintaining the model is in many cases difficult because of sensor limitation or imperfect. The plans produced by the planner often don’t give the effects in the real world that is anticipated. So planning cannot be done using this open-loop approach [Wyatt, 1997; Gat, 1998]. Brooks [1990a; 1990b; 1991a; 1991b] argues that this traditional approach is very
difficult, slow, and the resulting programs are very brittle and not generalizable to other robotic platform or to a different environment. It is clear that a different approach is required if one wants to bring robots to every day life. This was creating systems that are more robust that do not require too many modifications to the lay out of the environment where the robot will operate.

![Figure 2.1: Horizontal control architecture of the robot.](image)

To overcome these limitations behavior-based control approach is developed [Brooks, 1986]. In Behavior-based approach, instead of decomposing the task based on the functionality, the decomposition is done based on task-achieving modules or competences, are called behaviors on top of each other as shown in Figure 2.2, this is called vertical control architecture. The most common behavior-based control approaches are subsumption [Brooks, 1986] and motor schemas [Arkin, 1989; 1998; Arkin and Balch, 1997; Murphy, 2000] architectures, which made up of several parallel running behaviors. Each behavior calculates a mapping from sensor inputs - the sensor inputs relevant for the task of that behavior are used - to motor outputs (cf. Appendix A). The suggested motor outputs from the behavior with highest priority are used to control the robot's motors, or summed to generate one motors' output. These architectures are called behavior-based control approaches and represent methodologies for endowing robots with a collection of intelligent behaviors [Matarić, 1994b; 1995b; 1995c; 1997a; 1998a; Parker, 1996]. Behavior-based approaches are an extension of reactive architecture, their computation is not limited to lookup table and execution a simple

---

1 These are many infinitely man possible robot control schemes. These schemes are classified into four classes: reactive control ("don't think, react"), deliberative control ("think, then act"), hybrid control ("think and act independently in parallel"), and behavior-based control ("think the way you act"). Each of these approaches has its strengths and weaknesses, and all play important and successful roles in certain problems and application. The behavior-based and hybrid control have the same expressive and computational capabilities: both can store representation and look ahead. But they work in very different ways. Reactive control is also a popular in highly stochastic environments and demands very fast responses. Deliberative control is the best choice for domains that require a great deal of strategy and optimization, and in turn search and learning [Matarić, 2002]
functional mappings. Behavior-based systems are typically designed so the effects of the behaviors interact in the environment rather than internally through the system.

2.2 Subsumption Control Architecture

Brooks' subsumption architecture is a hierarchical architecture [Brook, 1986]. Lower levels control "instinctive" behaviors and higher-level control tasks that are regarded as more abstract. Each behavior consists of a fixed hardwired finite state machine (FSM), and cannot be altered without redesign the whole system as shown in Figure 2.3. The high-level behaviors can influence low-level ones through subsumption links. These links can either inhibit the lower levels completely, or influence their behaviors.

![Figure 2.2: Vertical control architecture of the robot.](image1)

![Figure 2.3: Behavior module as augmented FSA.](image2)
The large limitation of the subsumption architecture is its complexity. Creating correct subsuming and inhibitory links into other behaviors requires detailed knowledge of how the other modules, and the system as a whole, function. Another important limitation is the lack of explicit goals and its goal handling capabilities [Pirjanian, 1999]. Since there is no centralized control, there the higher-level behavior wins the arbitration when they compete against lower-level ones, and the functionality of subsumption architecture is viewed as an emergent property of the interaction of its behavior and depends on the static and dynamic properties of the environment.

2.3 Potential Fields

The potential-field approach is an approach to motion planning where the robot, which represented as a point in configuration space, move under the influence of an artificial potential field produced by an attractive force at the goal configuration and repulsive forces at the obstacles [Khateb, 1986]. The motor commands of the robot at any position in an artificial potential field correspond to the vector on which the robot is situated. Goal attract, and thus the goals will have vectors pointing towards them; obstacles repulse, and will be surrounded by vectors pointing away. Forces can be of constant magnitude on every vector in a system, or operate on a gradient. Many potential fields implementations have goal forces exert a constant force across the entire gradient, while obstacles will exert greater outward force the nearer the vector is to the obstacles. The vector at any given position is computed by summing all the forces exerted on that potential vector $U$. In a simple domain with one goal and two obstacles, the potential function $U$ is constructed as the sum of two more elementary potential functions:

$$U(q) = U_{att}(q) + U_{rep}(q)$$

(2.1)

Where $U_{att}$ is the attractive potential and $U_{rep}$ is a repulsive potential associated with obstacles. The force $\vec{F} = -\nabla U$ corresponds the most promising direction to move in. From Equation (2.1) it is seen that $\vec{F}$ is the sum of two forces: attractive $\vec{F}_{att} = -\nabla U_{att}$ and repulsive forces $\vec{F}_{rep} = -\nabla U_{rep}$. These forces vectors will generally point directly towards the goal except in the area around the obstacles as shown in Figure 2.4. There are many ways
to define the elementary functions. For example, the attractive potential field \( U_{at} \) can be defined as a parabolic well:

\[
U_{at}(q) = \frac{1}{2} \xi \| q - q_{goal} \|^2
\]  

(2.2)

Where \( \xi > 0 \) is a scaling factor and \( q_{goal} \) denotes the goal configuration. Figure 2.4(b) is a plot of \( U_{at} \) for a two dimensional cases. The repulsive potential field \( U_{rep} \) can define as follow:

\[
U_{rep}(q) = \begin{cases} 
\frac{1}{2} \eta \left( \frac{1}{\rho(q)} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q) \leq \rho_0 \\
0 & \text{if } \rho(q) > \rho_0
\end{cases}
\]  

(2.3)

Where \( \eta > 0 \) is a scaling factor, \( \rho(q) \) denotes the distance from \( q \) to the closest obstacle and \( \rho_0 > 0 \) is the distance within which the robot is under the influence of repulsive force. Figure 2.4(c) is a plot of \( U_{rep} \) for a two dimensional cases. Figure 2.4(d) is a plot of the total potential field function \( U \). These forces vectors will generally point directly towards the goal except in the area around the obstacles as shown in Figure 2.4.

Figure 2.4: (a) The configuration of obstacles (black) in the map. (b) The attractive potential associated with the goal configuration. (c) The repulsive potential associated with obstacles. (d) The sum of the attractive and repulsive potentials (adopted from Pirjanian, 1999).
There are two major difficulties with the potential fields method as applied to robot navigation. The first is that this method can be slow and require substantial internal state, as it appears that vectors must be computed based on a map of the entire region. But a potential fields method can be easily adapted to a reactive system. There is no need to factor in forces outside the of the robot’s perceptual range, though true potential fields required computing vectors based on the entire field. A goal outside the perceptual range may need to be maintained, but obstacles that the robot cannot sense do not necessarily need to affect the motion vectors. Thus the vector accompanying the current position of the robot can be computed using only the sensory information from a single time step [Arkin, 1989]. This vector computation using only current sensor readings makes this method reactive, in that it requires no internal state, and allows the vector on the current position of the robot to be computed quite quickly. The second potential field problem is \textit{local minima}. The robot’s motion at any moment is the sum of the forces affecting it. What happens, then when these forces sum perfectly to zero? The robot could remain indefinitely in a single position, perfectly immobilized at a local minimum. If no force disrupts this equilibrium, movement may cease. Thus any potential fields method must employ strategies for coping with this problem of local minima. One of these strategies is generating a time-varying noise vector.

\subsection*{2.4 Schema-Based Reactive Control}

The potential fields method is sufficient to create good behavior when the environment consists of only simple attractive and repulsive forces, but is not designed to support more complicated and situational-dependent forces. The motor schema method is the reactive control system components of Arkin’s Autonomous Robot Architecture (AuRA) and is an attempt to create a more general, behavior-based, conception of the \textit{potential field} method [Arkin, 1989; 1998; Arkin and Balch, 1997; Murphy, 2000]. A motor schema consists of several pre-programmed collection of parallel, concurrently active behaviors, some of which gather sensor information, called \textit{perceptual-schema} some derive effectors called \textit{motor-schema}. Each perceptual schema is responsible for one sensory modality, and each motor schema is responsible for one type of primitive behavior, expressing goal, or constraint for a task. The \textit{perceptual-schemas} directly feed into the motor schemas. These fields can be easily added to each other resulting in a field taking information from all behaviors into account. The direction and magnitude of the vector field are determined by the nature and gain of the
schema. The absence of arbitration between behaviors (schemas), the fusion of behavioral outputs using vector summation in a manner analogous to the potential field method.\(^2\)

As an example, important schema for a navigational task would include avoid_obstacles and move_to_goal. Since schemas are independent, they can run concurrently, provide parallelism and speed. Sensors inputs are processed by perceptual-schemas embedded in the motor behaviors. Perceptual processing is minimal and provides just information pertinent to the motor schema. For instance, a find_obstacles perceptual schema, which provides a list of sensed obstacles, is embedded in the avoid_obstacles motor schema.

The concurrently running motor schemas are integrated as follows. First, each scheme produces a vector indicating the direction the robot should move to satisfy that schema’s goal or constraint. The magnitude of the vector indicates the importance of achieving it. It is not critical, for instance, to avoid an obstacle if it is distant, but critical if close by. The magnitude of the avoid-obstacles vector is correspondingly small for distant obstacles and large for close ones. The importance of motor schemas relative to each other is indicated by the gain value for each one. The gain is usually set by human designer, but may also be determined by learning using genetic algorithms [Pearce et. al., 1992; Arkin, 1998]. Each motor vector is multiplied by associated gain value and the results are summed and normalized. The resultant vector is sent to the robot hardware for execution as shown in Figure 2.5.

Figure 2.5: Motor schema example. The diagram on the left shows a vector corresponding to move-to-goal schema; the center diagram shows avoid-obstacles schema. On the right, the two schemas are summed, resulting in a complete behavior for reaching the goal.

\(^2\) The programming of schema-based reactive control is demonstrated in Appendix A.
2.5 Temporal Sequencing

If a single set of schemata does not seem sufficient to obtain the desired behavior, motor-schemata can be clumped together into complex, emergent behaviors, in which a behavior consists of a number of different schemata. Groups of behaviors are referred to as behavioral assemblages. One way behavioral assemblages may be used in solving complex mission is to develop an assemblage for each task and to execute the assemblage in an appropriate sequence. The steps in the sequence are separate behavioral states. Perceptual events that cause transitions from one behavioral state to another are called perceptual triggers. The resulting mission-solving strategy can be represented as a Finite State Automaton (FSA). The technique is referred to as temporal sequencing [Arkin and MacKenzie, 1994; Arkin, 1998]. For example, the foraging task as shown in Figure 2.6, the forage task for a robot is to wander about the environment looking for items of interest (attractors). Upon encountering one of these attractors, the robot moves towards it, finally attaching itself. After attachment the robot navigation to the home base where it deposits the attractor. In this approach to solving the forage task, a robot can be in one of three behaviors states: wander, acquire and deliver. The robot begins in the wander state. If there are no attractors within the robot’s field of view, the robot remains in wander until one is encountered. When an attractor is encountered, a transition to the acquire state is triggered. While in the acquire state the robot moves towards the attractor and when it is sufficiently close, attaches to it. The last state, deliver, is triggered when the robot attaches to the attractor. While in the deliver state the robot carries the attractor back to home base. Upon reaching home base, the robot deposits the attractor there and reverts to the wander state [Balch, 1998a].

Figure 2.6: The Forage finite state automaton (FSA).
2.6 Motor Schema Versus Subsumption Architecture

Comparison with the subsumption architecture can highlight some distinctive aspects of the motor schema architecture. Subsumption architectures consist of a behavioral hierarchy, with each behavior running simultaneously and in parallel. As in motor schema approach, each behavior has an independent access to the sensory data, and uses that data to formulate an action to be taken, or it may decide to recommend no action at all. But unlike the motor schema approach, subsumption style architecture creates a hierarchy of behaviors, consisting of low-level behaviors that have no knowledge of the higher-level behaviors. Coordination of behaviors occurs according to the priority hierarchy in a *winner-take-all* approach. In the motor schema approach the action outputs of each of the schemes are *combined* by summation, and all the schemata are blended together to produce the actions of the robot at each time step (See §2.2 and §2.4). Each type of architecture has advantages and disadvantages. Subsumptive design gives a single behavior control over a system, meaning that the behavior is given complete priority. This can mean that the robot does whatever behavior is currently in control very well, but it can also mean that the robot loses sight of the larger goals of the system. Motor schemata allow the blending of behaviors, allowing the behavior of the robot to combine multiple goals. But motor schema architecture must contend with local minima and situations in which having *multiple goals* means that none of them is fulfilled to the designer’s satisfactions. Moreover, motor schemata inherent flexibility due to the dynamic *instantiation* and *deinstantiation* of behaviors on an as-needed basis, and easy *reconfigurability* through the use of high-level planners or *adaptive learning* systems [Pearce et al., 1992; Gat, 1998; Connell, 1991]. The comparison is summarized in Table 2.1.

<table>
<thead>
<tr>
<th></th>
<th>Subsumption architecture</th>
<th>Motor-Schemata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Reconfigurability/Learning</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Arbitration Mechanism</td>
<td>Competitive</td>
<td>Cooperative</td>
</tr>
<tr>
<td>Behavior implementation</td>
<td>Hardwired</td>
<td>Software</td>
</tr>
<tr>
<td>Behavior Reusability</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 2.1: Comparison between Subsumption architecture and Motor schema.*
2.7 Conclusion

The chapter shows that the behavior–based behavior-based control architectures introduce a new approach to robot control that overcomes the drawback in traditional approaches. Thus by decomposing the control into *task-achieving* modules called behaviors. The subsumption behavior-based architecture is a competitive arbitration scheme, where motor-schema architecture is a cooperative arbitration scheme, which is suitable for ours, since using the vector sum to compose behaviors is simple and intuitive. The pros and cons of the two most common used behavior-based control methods shows that the motor-schema technique is most suitable to implementation in our research.
CHAPTER 3

REINFORCEMENT LEARNING AND COOPERATION

The research effort into multi-robot system is driven by the assumption that multiple robots have advantages over single robot for the solution of some problems. We argue that that group(s) of cooperating intelligent robots can be more flexible, more reliable and more fault-tolerance, and more economical than a single, monolithic robot only and only if efficient means can be found for cooperating robot group(s). The thesis focuses on cooperative robotics research from the organization of the groups and the reinforcement learning perspectives. The reinforcement learning algorithms, which will suit for learning in, robots are introduced. Starting by presenting the basic reinforcement-learning framework and then discuss a series of popular reinforcement learning algorithms and techniques in this context. These techniques are used cast in Markov decision processes, and are mainly applicable where a dynamic programming solution is not available. Moreover, other reinforcement techniques, which are not cast in Markov decision processes are presented, and a comparison between them to show their pros and cons. In addition, the previous research approaches of learning multiple robots cooperation were presented, in attempt to motivate research into learning dynamic organization. The goal is to establish a base from which the research proceeds and to differentiate it from similar work.

3.1 Introduction

Certain approaches have proven quite successful and have caused a re-analysis within the field of artificial intelligence of what components are necessary in the intellectual architecture of such multi-robot systems. In order to built these multi-robots systems to come close for that the people dream with them. Researches suggested that embedding learning techniques in multi-robot to incite them for efficiency and cooperation. Since that, the learning has often been neglected, and the designer of multi-robot systems have generally taken on the extremely difficult task of trying to anticipate all possible contingencies and interactions among the robots ahead of time. The machine learning techniques ultimate goals are to deal with the noise in their internal and external sensors and the inconsistencies in the behavior of the environment and other robots [Parker 1994; 2000a; Matarić 1995a; 1998b; 2001].
One these learning approach is the reinforcement learning (RL) [Kaelbling at. el., 1996; Sutton and Barto, 1998; Keerthi and Ravindran, 1999; Holland, 1986; Goldberg, 1989; Michalewicz, 1994; Koza, 1992] which has emerged as an adaptive way in an elegant way can acquires collective behavior experience from multi-robots environments and attempt to direct the robots behavior away from individual greediness and towards global efficiency.

3.2 Reinforcement Learning

The artificial intelligence, a very general model of environment is assumed to be deterministic and by planning and heuristic search typically attempt to construct open-loop control policies. Such methods are very general in that they can find correct policies for complex, non-linear environment if the model is accurate, perception completes, and controlled system is deterministic. In complex dynamic environments for which we don’t posses models, and in which the robot may be hampered by incomplete perception and unreliable effectors. On the other hand, control theory gives us method for constructing closed-loop adaptive control policies that depends on the environment’s state and the time [Singh 1994; Barto et. al., 1995]. Techniques for constructing such control policies include analytical methods for simple (linear) systems, and numerical methods such as dynamic programming for more complex (non-linear and stochastic) systems. These methods are for constructing control policies off-line, given accurate environment model [Wyatt, 1997]. When it is trying to design adaptive control policies for robots we typically have no model of the environment. Thus the machine learning arises for developing on-line and model-free approaches and algorithms for constructing robot’s closed-loop policy by automatically acquiring knowledge, developing strategies and modifying its behavioral tendency by experience [Kaelbling at. el., 1996; Sutton and Barto, 1998; Keerthi and Ravindran, 1999; Mahadevan 1996b; Russell 1996; Mitchell, 1997; Dietterich 1997]. Reinforcement learning (RL) which describes a large class of learning tasks and algorithms characterized by the fact that the robot learns an associative mapping, $\pi = X \rightarrow A$ (from state space $X$ to the action space $A$) by maximizing a scalar evaluation (rewards) of its performance form the environment. The robot learns strategies for interacting with its environment in closed-loop cycle as shown in Figure 3.1. This is the basic reinforcement-learning model where the robot environment interaction is viewed as a coupled dynamic system in which the outputs of the robot are transferred into inputs to the environment and the outputs of the environment are transformed into inputs to the robot. The inputs to the robot usually do not capture a complete description of the environment’s
state and the environment changes continuously through time. Furthermore, the robot influences the future sequence of its inputs by its outputs at each stage. Through that, the robot explores its dynamic environment, attempting to monitor the environment and figure out by itself how it can exploit acquired experience to maximize its reward, which will receive based on the sequences of the actions it takes.

Learning solely from reinforcement signals, makes the environment a powerful force shaping the behaviors of autonomous robots. Robots can also directly shape their environments. Because of this interaction of shaping forces, it is important to develop autonomous robots in a system where all of the forces for change are learnable to optimize system performance over its lifetime. When learning control policies, we must be able to evaluate them with respect to each other. The robot’s goal is to find, based on its experience with the environment, a strategy, or an optimal policy, \( \pi \), for choosing actions that will yield as much reward as possible. The most obvious metric, the sum of all rewards over the life of the robot,

\[
\sum_{t=0}^{\infty} r_t, \quad (3.1)
\]

is generally not used. For robots with infinite lifetimes, all possible sequences of rewards would sum to infinity. This is not the case for robot with a finite lifetime, however. In this case, the obvious metric turns into the finite horizon measure,

\[
\sum_{t=0}^{k} r_t, \quad (3.2)
\]
This measure sums the rewards over some finite number, \( k \), of time steps. Average case,

\[
\lim_{k \to \infty} \frac{1}{k} \sum_{t=0}^{k} r_t. \tag{3.3}
\]

Extends this by using the average reward received over the whole lifetime of the robot. The infinite horizon discounted measure,

\[
\sum_{t=0}^{\infty} \gamma^t r_t, \tag{3.4}
\]

Uses a discount factor, \( 0 \leq \gamma \leq 1 \), to control the relative importance of short-term and long-term reinforcements. As \( \gamma \to 0 \) the short-term reinforcement signals became more important. When \( \gamma = 0 \) the only reinforcement signal that matters is the immediate reinforcement signal, and we obtain the one-step greedy policy; the best action is the one that gives the greatest immediate reward. The optimal finite horizon policy may be non-stationary in the sense that different actions may be selected for the same state at different time steps. In problems with hard time lines, finite horizon policies are the right choice. In problem without hard deadlines, \( k = \infty \), and an infinite-horizon policy is desired. It is theorem that the optimal infinite horizon policy is stationary [Haykin, 1999]. Most of the reinforcement-learning problems are typically cast as Markov decision processes (MDPs)\(^3\) as shown in Figure 3.2.

![Figure 3.2: The Markov decision process model.](image)

In Markov decision process there is a finite set of states, \( S \), a finite set of actions, \( A \), and time is discrete. The reward function

\[
R : S \times A \to \mathbb{R} \tag{3.5}
\]

---

\(^3\)Complete observability of the state is necessary for learning methods based on MDPs. In the case of noisy and incomplete information for the robot's observations, the "perceptual aliasing" or "hidden state" problem arise. So, the MDP framework needs to extend by partially observable Markov decision process (POMDP). See Kaelbling et al. (1996) for the strategies implemented to deal with this problem.
returns an immediate measure of how good an action was. The resulting state, \( s_{t+1} \), is
dependent on the transition function

\[
T : S \times A \rightarrow \Pi(S)
\]  
(3.6)

Which returns a probability distribution over possible next states. An important property of
MDPs is that these state transitions depend only on the last state and action. This is known
as the Markov property. The problem, then, is to generate a policy, \( \pi : S \rightarrow A \), based on these
immediate rewards that maximize our expected long-term reward measure such as:

\[
V^*(s_0) = E\left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \right].
\]  
(3.7)

Where \( E \) indicates expected values, and \( R(s, a) = \sum_s T(s, a, s')R(s, a, s') \) is the expected
reward for state-action pair. If we know the functions \( T \) and \( R \) then we can define an
optimal value function, \( V^*(s) \), over states:

\[
V^*(s) = \max_a \left[ R(s, a) + \gamma \sum_{s'} T(s, a, s')V^*(s') \right].
\]  
(3.8)

This function assigns a value to each state, which is the best immediate reward that we can
get for any action from that state added to the optimal value from each of the possible
resulting states, weighted by their probability. If we know this function, then we can define
the optimal policy, \( \pi^* \) by simply selecting the action, \( a \), that gives the maximum value:

\[
\pi^*(s) = \arg \max_a \left[ R(s, a) + \gamma \sum_{s'} T(s, a, s')V^*(s') \right].
\]  
(3.9)

Equation (3.9) is called Bellman’s optimality equation. It provides a mean of computing the
optimal policy for the Markov decision process by performing one-step look-ahead search
and choosing the action whose backup value is the largest. There are well-understood
dynamic programming based methods for computing \( V^* \) such as value iteration and policy
iteration, which lead to a simple procedure for learning the optima value function, and hence
the optimal policy. The two algorithms, policy iteration and value iteration, are off-line and
model-based methods, they need to learn \( T \) and \( R \) functions [Dietterich, 1997]. Instead of
learning $T$ and $R$, however we can incrementally learn the optimal value function directly. In the next sections, we describe well-known algorithms that attempt to iteratively approximate the optimal value function.

### 3.3 Monte Carlo Learning

Dynamic programming methods require complete environment dynamics to find the optimal value function. However, in problems domains, full environmental information is not available, or there is no model of the environment. For such cases, environment must be observed through experience. Monte Carlo methods provide some means of making this possible. Monte Carlo methods are for *episodic* tasks only, because they are based on complete averaged returns. By episodic we mean that task somehow (with success or failure) terminates and then restarts. MC methods learn episode-by-episode manner, rather than step-by-step manner. This implies that the value function estimates and policies are changed after each episode and remain same until the episode complete. Monte Carlo methods average the values of the states that are visited in an episode. After many episodes and many visits to each state, the averaged values became approximately correct estimations.

#### First-visit Monte Carlo Algorithm

<table>
<thead>
<tr>
<th>Initialize</th>
<th>$\pi \leftarrow$ Policy to be evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V(s) \leftarrow$ an arbitrary state-value function.</td>
<td></td>
</tr>
<tr>
<td>$\text{Returns}(s) \leftarrow$ an empty list, for all $s \in S$</td>
<td></td>
</tr>
</tbody>
</table>

Do forever:

- Generate an episode using $\pi$.
- For each state $s$ appearing in the episode:
  - $R \leftarrow$ return following the first occurrence of $s$.
  - Append $R$ to $\text{Returns}(s)$
  - $V'(s) \leftarrow$ average ($\text{Returns}(s)$)

End Do

Table 3.1: First-visit Monte Carlo method for estimating $V^\pi(s)$.

In order to estimate $V^\pi(s)$, the value of state $s$ under policy $\pi$, given a set of episodes obtained by following $\pi$ and passing through $s$, we first define each occurrence of state $s$ in an episode to be a *visit* to $s$. Then, *every-visit Monte Carlo method* estimates $V^\pi(s)$ as the average of the returns following all the visits to $s$ in a set of episodes. Another approach, called *first-
visit Monte Carlo method (as shown in Table 3.1) averages just the returns following first visit to \( s \). Both first-visit and every-visit Monte Carlo converge to \( V^\pi(s) \) as the number of visits (or first visits) to \( s \) goes to infinity [Kaelbling et al., 1996; Sutton and Barto, 1998; Keerthi and Ravindran, 1999].

### 3.4 Temporal Difference Learning

**Temporal difference learning (TD (0))** [Sutton 1988] is a combination of Monte Carlo and dynamic programming methods for learning a value function for states, \( V(s) \), based on state transitions and rewards, \((s_t, r_{t+1}, s_{t+1})\). Starting with a random value for each state, it iteratively updates the value-function approximately according to the following update rule:

\[
V(s_t) = V(s_t) + \alpha(s_t)[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
\] (3.10)

There are two parameters; a learning rate, \( \alpha \), and a discount factor, \( \gamma \). The learning rate controls how much we change the current estimates of \( V(s) \) based on each new experience.

Equation (3.10) forms the basis for the TD (0) algorithm which is really an instance of a more general class of algorithms called TD \((\lambda)\), with \( \lambda = 0 \), where \( 0 \leq \lambda < 1 \) is a parameter that controls how much of the current difference \([r + \gamma V_{t-1}(s_{t+1}) - V_{t-1}(s_t)]\) is applied to all previous states. As shown in Table 3.2, TD (0) looks only one step ahead when adjusting value estimates; although it will eventually arrive at correct answers, it can take quite a while to do so.

The general TD \((\lambda)\) is similar to TD (0) but is generalized to propagate information across trajectories of states according to its eligibility, \( e_t(s_t) \), which is a function of \( \lambda \) that indicates how far in the past and how frequently the state has been visited.

\[
V(s_t) = V(s_t) + \alpha(s_t)[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]e_t(s_t)
\] (3.11)

The value of \( e_t(s_t) \) is updated each transition for all \( s \in S \). There are two forms of update equations. An **accumulating trace** updates the eligibility of a state using,

\[
e_t(s) = \begin{cases} 
\gamma \lambda e_{t-1}(s) + 1 & \text{if } s = \text{current state} \\
\gamma \lambda e_{t-1}(s) & \text{otherwise}
\end{cases}
\] (3.12)

A **replacing trace** update is defined by,
Under both mechanisms, the eligibility of a state decays away exponentially when the state is unvisited. Under an accumulating trace, the eligibility is increased by a constant every time the state is visited and under a replacing trace, eligibility of a state is reset to a constant on each visit. An eligibility trace can be thought of as a short-term memory process, initiated the first time a state is visited by a robot. The degree of activation depends on the recency of the most recent visit and on the frequency of visits. Thus, eligibility traces implement two heuristics, a recency heuristic, and a frequency heuristic [Sutton and Barto, 1998]. Sutton [1988] and others [Dayan, 1992; Dayan and Sejnowski, 1994] note that TD (λ) converges more quickly for $\lambda \neq 0$. Learning the value function $V(s)$ for a fixed policy can be combined with a policy-learner to get what is known as an actor-critic or an adaptive heuristic critic system [Barto et. al., 1983]. This alternates between learning the value function for the current policy, and modify the policy based on the learned value function.

$$e_t(s) = \begin{cases} \gamma^\lambda e_{t-1}(s) & \text{if } s = \text{current state} \\ 1 & \text{otherwise} \end{cases}$$

\text{(3.13)}

### 3.5 Q-learning Algorithm

Instead of maintaining both a policy and value function using actor-critic techniques, Q-learning [Watkins, 1989] performs essentially the same function by combines them into a single function, known as the $Q$-function. This function, $Q(s, a)$, reflects how good it is, in a long-term sense that depends on the evaluation measure, to take action $a$ from states. On other words, the quantity $Q(s, a)$ gives the expected cumulative discounted reward of
performing action \( a \) in state \( s \) and then pursuing the current policy thereafter. It uses 4-tuples \( (s_t, a_t, r_{t+1}, s_{t+1}) \) to iteratively update an approximation to the optimal \( Q \)-function,

\[
Q'(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_a Q^*(s', a').
\]

(3.14)

Once the optimal value function is known, the optimal policy, \( \pi^*(s) \) can be easily calculated:

\[
\pi^*(s) = \arg \max_a Q^*(s, a).
\]

(3.15)

Starting with a random the \( Q \)-function can be approximated incrementally according the following rule:

\[
Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[ r_t(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a') - Q(s_t, a_t) \right]
\]

(3.16)

The Q-learning algorithm is summarized in Table 3.3. Watkins and Dayan [1992] proved the iteration converges to \( Q^*(s, a) \) under the condition that the learning set includes an infinite number of episodes for each state and action. If the function is properly computed, an agent can act optimally simply by looking up the best-valued action for any situation. The Q-learning chooses the optimal action for state based on the value of the \( Q \)-function for that state. If only the best action is chosen, it is possible that some actions will never be chosen from some state. This is bad, because we can never be certain of having found the best action from a state unless we have tried them all. So, we must have sort of exploration strategy, which encourages us to take some non-optimal actions in order to gain information about the world. This is known as the exploration/exportation problem [Thrun, 1992a, 1992b; Wiering, 1999].

<table>
<thead>
<tr>
<th>Q-learning Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each pair ((s, a) \in S \times A), initiate the table entry ( Q(s, a) ) to 0.</td>
</tr>
<tr>
<td>Observe the current state ( s ).</td>
</tr>
<tr>
<td>Do forever:</td>
</tr>
<tr>
<td>- Select an action ( a ) and execute it.</td>
</tr>
<tr>
<td>- Receive immediate reward ( r_{t+1} ).</td>
</tr>
<tr>
<td>- Observe the new state ( s_{t+1} ).</td>
</tr>
<tr>
<td>- Update the table entry using real-experience for ( Q(s_t, a_t) ) as Equation (3.16).</td>
</tr>
<tr>
<td>- Set ( s ) to ( s_{t+1} ).</td>
</tr>
<tr>
<td>End Do</td>
</tr>
</tbody>
</table>

Table 3.3: Q-learning algorithm.
The simplest exploration strategy follows a policy of *optimism in the face of uncertainty*. When the effect of an action is poorly understood, then it is usually better to take that action in order to reduce this uncertainty. For example, introducing and propagating *exploration bonuses* to artificially raise the estimated value of untried actions [Sutton, 1990a; Wyatt, 1997; Wiering, 1999]. Another approach is \( \epsilon \)-greedy strategy, is to select a random action a certain percentage of time \( \epsilon \), and the best action otherwise [Sutton and Barto, 1998]. The last approach attempts to take into account how certain we are that are god or bad. It is based on the Boltzmann or Gibbs distribution and is often called *soft-max* exploration. Where at each time step, action \( a \) is chosen from state \( s \) with the following probability:

\[
\Pr(a | s) = \frac{e^{Q(s,a)/T}}{\sum_a e^{Q(s,a)/T}}
\] (3.17)

The parameter \( T > 0 \) is called the temperature. Varying the temperature parameter provides a continuum of stochastic policies, varying between the random policy at height temperature and greedy policy at low temperature. Even at fixed temperature, the nature of the policy will change as the system learns, because a lot of exploration occurs in states where \( Q \)-values for different actions are almost equal, and little exploration in states where \( Q \)-values are very different.

### 3.6 SARSA Learning Algorithm

While \( Q \)-learning update \( Q \)-values by making use of an assumption that the currently best action will be taken at the next step, there is a simple scheme which makes use of the real-world experience instead. This approach is called SARSA, which is introduced by Rummery and Niranjan [1994]. SARSA is similar to \( Q \)-learning in that it attempts to learn the state-action value function \( Q(s,a) \). The main difference between SARSA and \( Q \)-learning, however, is in the incremental update function. Actually, the \( \max \) operator is dropped from update equation and state action value of the next state-action is used. The update equation is:

\[
Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_t(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
\] (3.18)

Just like TD, SARSA learns the value for a fixed policy and must be combined with policy learning component in order to make a complete reinforcement system.
3.7 Which Algorithm Should We Use?

The algorithms presented in previous sections have all been shown to be effective in solving a variety of reinforcement learning tasks. However, there is a fundamental difference between Q-learning and the other that makes it much more appealing for our purposes. The temporal difference and SARSA algorithms are known as on-policy algorithms. The value function that they learn is dependent on the policy that is followed during learning. Q-learning on the other hand, is an off-policy algorithm. The learned policy is independent of the policy followed during learning. Using an off-policy algorithm, such as Q-learning, frees us from worrying about the quality of the policy that we are follow during the learning, since we might not know a good policy for the task that we are attempting to learn. Using an on-policy algorithm with an arbitrary bad training policy might cause us not to learn the optimal policy. Using an off-policy method allows us to avoid this problem by using exploration strategies to maximize the information obtained during learning [Smart, 2002]. There are still potential problems, however, with the speed of convergence to the optimal policy when using training policies that are not necessarily very good.

3.8 Planning in Reinforcement Learning

Reinforcement learning algorithms learn from experiences obtained interacting with the environment. Although making errors are required to learn, they may cost robot’s life (Hazards situations). If the robot has an almost correct model, it can produce can produce hypothetical experience to learn from them, since the learning process for an optimal policy needs many experiences [Lin, 1992; 1993; Mahadevan and Connell, 1992]. From an AI point of view, the optimal policy might obtained if the system had carried out explicit planning, which can be thought as learning the possible future robot-environment interactions, from the con-specifics to determine long-range consequences. Planning is learning from these hypothetical experiences and useful when making errors are costly, robot has less real experiences from the interaction with the environment, and speeding up the learning process is urgently [Sutton, 1990a; 1990b; 1991; Clouse, 1997]. The earliest use of planning in the reinforcement learning was the Dyna architecture as shown in Table (3.4) that integrates learning and planning which proposed by Sutton’s [1990a; 1990b; 1991]. The planning is treated as being virtually identical to reinforcement learning except that while learning updates the appropriate value function estimates according to real experience, planning differs only in that updates these same value
function estimated for hypothetical experience chosen from the world model. The model component of Dyna algorithm is a table indexed by state action pairs and contains the resultant state and a reward as an entry [Sutton and Barto, 1998]. The table is filled while real experience gathered. When state and action space are too large, storing all experience become impractical. There is a version of Dyna called Prioritized Sweeping [Moore and Atkeson, 1993] and closely related Queue-Dyna [Peng and Williams, 1993] that store fixed number of most useful experiences and updates experience tuples for which there will be large update errors using prioritize value functions. Lin [Lin, 1993] proposed the experience replay method, where the past useful experience are stored in a fixed length list and presented to the reinforcement learning again and again in reverse order. This is also a kind of planning because some costly experiences from the past are used to learn from once again. This is useful for learning in real worlds for which exploring new experiences is often much more expensive than replaying old experiences.

<table>
<thead>
<tr>
<th>Q-Dyna-Learning Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each pair ((s, a)), initiate the table entry (Q(s, a)) to 0.</td>
</tr>
<tr>
<td>Observe the current state (s_t).</td>
</tr>
<tr>
<td>Do forever:</td>
</tr>
<tr>
<td>- Select an action (a_t) and execute it.</td>
</tr>
<tr>
<td>- Receive immediate reward (r_{t+1}).</td>
</tr>
<tr>
<td>- Observe the new state (s_{t+1}).</td>
</tr>
<tr>
<td>- Update the table entry using real-experience for (Q(s_t, a_t)) as Equation (3.16).</td>
</tr>
<tr>
<td>- Set (s_t) to (s_{t+1}).</td>
</tr>
<tr>
<td>- Update Model using this real experience.</td>
</tr>
<tr>
<td>Repeat (k) times</td>
</tr>
<tr>
<td>- Predict reward (r_{t+1}) and next state (s_{t+1}) from the Model</td>
</tr>
<tr>
<td>- Update the table entry using hypothetical experience for (Q(s_t, a_t)) as Equation (3.16).</td>
</tr>
<tr>
<td>End Repeat</td>
</tr>
<tr>
<td>End Do</td>
</tr>
</tbody>
</table>

Table 3.4: Q-Dyna learning algorithm.

3.9 Designing the Reinforcement Functions

Many researchers recognized that for any real artificial agent implementation, there is no convenient reinforcement function in the world inserting appropriate values into the robot [Karlsson, 1997]. The reinforcement function is thus depicted as a part of the robot that translates the observed state into a reinforcement value. That is, part of the robot's a priori
knowledge is the ability to recognize states and assign a value to them. Balch [1998b] provides taxonomies for various kinds of reinforcement functions that have developed for multi-robot systems based on numbers of issues. The taxonomy of multi-robot reinforcement functions is to examine the source of reward, relation to performance, locality, time and continuity. In order to define the reinforcement function, the designer distills a large amount of task specific knowledge into a number. Thus, the reinforcement function usually closely coupled to the performance metric for a task. Since learning robots strive to maximize the reward signal provided them, performance is maximized when their reward closely parallels performance. It is sometimes the case however, in many embedded autonomous agent applications, especially in multi-robot systems, robots cannot or should not be rewarded strictly according to overall system performance. For example, the reinforcement function designer will encode a notation of what classes of states represents the robot’s goal. However, the robot’s sensors do not provide enough information for an accurate computation of performance, reward is delayed or performance depends on the actions of other robots over which the agent has limited knowledge and/or control [Balch, 1998b]. Since the reinforcement function is the robot’s only indication of how well it is meeting its goals, it must be designed with great care. For all practical purposes, the reinforcement function defines the robot’s task. For complex problems, it may be necessary to refine the reinforcement function as a robot discovers ways to get the reinforcement without accomplishing the task. In order to design a correct a reinforcement function, there must first be a clear idea of what the robot’s goals should be and the reinforcement function parallels the performance metric [Balch, 1998b]. Given a simple goal, it is straightforward to define a reinforcement function leading to the proper behavior. The reinforcement learning gains much of its power from the ability to use reinforcement function to specify different degrees of goal satisfaction, and identify sub-goals of a complex task. Rewarding sub-goals is one way to encode domain, task knowledge, and speedup learning [Matarić, 1994c]. The relative magnitude of reinforcements needs to reflect the relative importance of the states being rewarded. One could prioritize sub-goals by assigning reinforcements of greater magnitude to the more important ones [Karlsson, 1997].

3.10 Other Reinforcement Learning Techniques

There are other two reinforcement-learning techniques that are not cast in Markov decision process framework. The two techniques take differing approaches, with disparate
assumptions, and algorithms to endow the robot with the capacity to learn to solve problems. These techniques that we present in this section are: (1) evolutionary and (2) statistical reinforcement learning [Holland, 1986; Goldberg, 1989; Moriarty et al., 1999; Maes and Brooks, 1991; Parker, 1994, 1995b; 1998a; 2000b].

3.10.1 Evolutionary Reinforcement Learning

Evolutionary learning algorithms [Goldberg, 1989; Michalewicz 1994; Koza, 1992] are used to search the space of decision policies, and use a direct mapping that associates state descriptions directly with a recommended action. Evolutionary algorithms encode the potential policies into structures, called chromosomes. During each iteration, the evolutionary algorithms evaluate policies and generate offspring based on the fitness of each solution in the task. Substructures, or genes, of the policy are then modified through genetic operators such as mutation and crossover. The idea is that policies that lead to good solutions in previous evaluations can be mutated or combined to form even better policies in subsequent evaluations. The policies evaluated by a so-called fitness or objective function, indicating its utility or immediate rewards as of a candidate policy [Moriarty et al., 1999]. Several researchers employed an evolutionary reinforcement learning, which combines genetic evolution with rich variety of structures that been but under evolution like neural networks [Lund 1995; Floreano 1997], symbolic rule-based [Grefenstette and Schultz 1994], and fuzzy rules [Hoffmann, 1998] for robot control. The evolutionary learning algorithms are off-on and model-free approaches, however they address the credit assignment and statistics information implicitly based on the fitness evaluations of the entire policy sequence obtain [Moriarty et al. 1999]. In addition, evolutionary learning algorithms required a lot of memory space and evaluation time for the policies, which make them actually off-line learning techniques [Matračić and Cliff 1996]. The co-evolution has been used for evolving robust cooperative and competitive behavior in societies, but has not yet been applied to evolve control policies for physical robots [Matračić and Cliff 1996].

---

4 Evolutionary learning is a form of learning. The main difference between evolution and learning is concerned with how the adaptation process is carried out in space and time. Evolution is based on a population of individuals, but learning taking place within a single individual. Evolution capture relatively slow environmental changes on the other hand, learning capture environmental changes faster.

5 The combining of the evolution and learning during the life of the robot is one way to overcome the lengthy evolutionary time scale. The learning can improves the search properties of artificial evolution by making the controller more robust to changes occur very fast. This is typically achieved by evolving neural controllers that can learn with a reinforcement learning or back-propagation.
3.10.2 Statistical Reinforcement Learning

Statistical learning is deals with learning a world model using analysis and interpretation of data. These world models are usually not known a priori, it is possible to learn the world model from experiences. In statistical reinforcement algorithms - by using of a mathematical world model combining a set of parameters - the robot try to learn the parameters adjustments or altering the precondition list for the behaviors activation using the collected data through monitoring the performance of the system through accumulating a statistical factors. Such that correlation between the reinforcement signal and behavior activation [Maes and Brooks, 1991], the average, and standard deviations of the task performance [Parker, 1994; 1995b; 2000b]. These mathematical world models guided the system to learn the conditions of activation and selection of the behaviors through feedback signals. Furthermore, they are useful to explain causal relationship, to explain why the robot prefers some choice over another, and to report what the robot has been doing most of its time.

3.11 Comparison Between Learning Techniques

One can characterize the techniques along dimensions such as: the memory space used by these algorithms, tracking changing world and noise, world-model is learned for an optimal policy, and the optimality of the learned policy. One can also discuss how each technique deals with the credit assignment and explore/exploit problems. Finally, another quality on which the methods differ is the speed with which each achieves its objective. One can compare and contrast these techniques along these dimensions and notice their pros and cons. Learning to solve a problem needs to find a policy. Each algorithm uses different data structure and memory allocation. In evolutionary learning algorithms a lot of memory space and evaluation time is required for the policies, they create a population of candidate policies to be evaluated, which make them actually off-line learning techniques [Matrač and Cliff 1996]. The other techniques, they reserved only one policy and gradually improve it. Moreover, this enables them to track changing world and noise. The evolutionary learning algorithms are independent of any model of the environment in learning the policy, the temporal difference can be either model-free or model-based approach, and the statistical approach is based completely on a world-model. The using of the world models speed up learning process and reaching the objectives. In addition, world models enable to track changing world and noise. The optimality of learned policy is also a characteristic of concern. There is no evidence that a
Robot employ evolutionary learning will learn an optimal policy, especially with premature convergence [Goldberg 1989; Michalewicz 1994]. For statistical learning, the barely have empirical evidence [Maes and Brooks, 1991; Parker, 1994; 1995b] that the learner will develop a useful policy at all, let alone an optimal one. On the other hand, there is much theoretical support for the claim that temporal reinforcement learning robot will eventually learn an optimal policy. Many convergence proofs for TD (λ) and Q-learning exist [Watkins 1989; Watkins and Dayan 1992; Tsitsiklis 1994; Sutton and Barto, 1998]. Reinforcement learning must face the credit assignment problem and the explore/exploit tradeoff. Learning with an evolutionary learning algorithms address the credit assignment and statistics information implicitly based on the fitness evaluations of the entire policy sequence obtain [Moriarty et al., 1999], with one exception for the classifier system. Temporal difference learning addresses the credit assignment and statistics information explicitly by distribution of credit and statistic for every state. On the other hand, statistical reinforcement learning, address credit assignment explicitly by correlating samples experience with reinforcement rewards. Finally, both of evolutionary learning and temporal difference algorithms needs to explore their environment, but for statistical learning, there is no reason for exploring since the experience data is cataloged and correlated with reinforcement signals. The comparison is summarized in Table 3.5. In our comparison, we point out why the temporal difference and statistical learning techniques are preferred in the robotics and multi-robot domains [EL-Telbany et al., 2002b].

<table>
<thead>
<tr>
<th></th>
<th>Evolutionary learning</th>
<th>Temporal Difference</th>
<th>Statistical learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory space</td>
<td>Large</td>
<td>Small</td>
<td>Small</td>
</tr>
<tr>
<td>Tracking changes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Model-based learning</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Learning speed</td>
<td>Slow</td>
<td>Middle</td>
<td>Fast</td>
</tr>
<tr>
<td>Optimality of policy</td>
<td>Mostly</td>
<td>Approached</td>
<td>Mostly</td>
</tr>
<tr>
<td>Credit assignment</td>
<td>Implicit</td>
<td>Explicit</td>
<td>Explicit</td>
</tr>
<tr>
<td>Explore/exploit problem</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.5: Comparison between different reinforcement techniques in a set of characteristics.
3.12 Cooperation in Multi-Robot Systems

Cooperation is a vital part of applying multiple robots to a mission in an efficient manner. Cooperation is a form of interactions among robots, is relatively new, and has only recently begun to be particularly studied by artificial intelligence community [Cao et al, 1997]. This leads to a significant prior research in multi-robot cooperation, and generate two main approaches have been dubbed collective robotics and cooperative robotics. Both approach the same problem from a different angle- how to obtain a desired group behavior from a multi-robot system by engineering the behavior of individuals. Collective robotics, also known as swarm robotics, can be characterized by distributed control of homogenous robot teams [Bonabeau et al., 1999]. The robots do not explicitly work together, but desired cooperative behavior is obtained as emergent property form their interactions with each other and the environment. The types of tasks implemented take inspiration from eusocial insect societies, such as ants [Kube and Zhang, 1992; 1993; 1996; Kube and Bonabeau, 1998; Bonabeau et al., 1999]. So, emergent cooperative behavior has traditionally been applied to homogenous robot groups and has relied heavily on redundant skills across the group to achieve good overall performance. Cooperative robotics covers the rest of the field. It is involves systems of heterogeneous robots explicitly and intentionally cooperate with a purpose often achieved by planning. The cooperation in multi-robot systems need a certain amount of extra-work (i.e., mechanism of cooperation), and leads to increase of performance or saving time [Parker 1992; 1994; 1996; 1998b; Matarić et al., 1995; Matarić, 1997b; Yanco and Stein, 1993; Doran et al., 1997]. This brings us to the following definition:

**Definition 3.1:** Cooperation is a certain amount of extra-work (i.e., mechanism of cooperation), and leads to increase of performance or saving time.

This extra-work can be achieved by three methods, those based on communication, those based on convention (social rules) and those based on learning [Cao et al, 1997].

- **Communication** is static method of cooperation lie in the imposition by the designer of communication structure in the interaction dynamics of multi-robot behaviors. Communication is necessary for cooperation, but not sufficient. That is, any system that cooperates must communicate, but not all systems that communicate are cooperating [Kaiser et al., 1996; Doran at. el., 1997; Matarić et al., 1995; Matarić, 1997b]. In order to cooperate, robots must first communicate with each other, and then secondly the must use the information that they communicate to better their performance as an individuals and/or as a
group. Arkin [1992], Arkin et al., [1993] and Balch and Arkin [1994] attempt to answer the question of whether explicit communication is useful in shared-goal environment. Arkin et al. examines a simple forage task under varying level of communication. Since, in this simulation, multiple robots can cooperatively carry an object toward their goal much faster than a single robot can, there is a distinct advantage to cooperating. With no communication, Arkin et al.'s robots simply wander until they find an object, then pick it up and bringing it back to their goals. When the robot can communicate, however, a robot that finds an object broadcasts its location to other robots, which then attempt to assist it in carrying the object back to its goal. Simulation found a moderate decrease in distance traveled per robot in the cooperative scenario, a large decrease in the number of steps needed to carry an object toward home. Arkin et al. argues that, because the distance traveled is an average over number of robots, the fact that step-to-goal is decreasing is an indication that robots are doing more useful work. Parker [1994; 1995a; 1998a] showed that mobile robots that explicitly communicated with each other performed a task more efficiently than those that did not. In contrast, Balch and Arkin [1994] describe a situations where communication is not necessary helpful to cooperating robots. Balch and Arkin investigate three multi-robot tasks. Foraging, in which robots search for and retrieve goal objects in an arena. Consuming, in which robots find and operate on goal objects in the place where they are found. Grazing, in which robots’ paths must completely cover the space of arena. Foraging is similar to garbage collection mission; grazing is similar to a repair mission, and grazing is similar to lawn-moving or flat cleaning mission. Balch and Arkin experiments with three levels of communication: no direct communication, state communication, and goal communication. State communication refers to the case when agents broadcast their internal state. For example, robots broadcast the fact that they are wandering aimlessly in state communication. Goal communication is the case where robots broadcast information related to a goal – for example, the location of a goal object in a foraging mission. Balch and Arkin find that the type of task to be performed greatly affects the performance of the various communication schemes. Under both forage and consume, state and goal communication are an improvement over no communication. However, goal communication proves better under the forage mission, while state communication proves better under the consume mission. Surprisingly, neither goal nor state communication significantly improves performance under the grazing task. Balch and Arkin attribute this fact to the presence of implicit communication where there was no intention on the part of the sender to communicate. As robot graze, they
leave behind them a record of their actions. Other robots do not need explicit messages from acting agent, as they are able to infer what they need by observing the state of the world. So, the environment provide an implicate communication channels, which can be used to provide awareness of one agent by another. For example, using vision as a framework for cooperation by observation.

- **Conventions** (social rules) are static methods of cooperation lie in the imposition by the designer of a control structure in the interaction dynamics of multi-robot behaviors. This by reducing the burden on a robot of recognizing the actions of the others’ by installing knowledge of how others behave in particular situations. Tennenholtz and Shoham [1992] investigated mechanisms for realizing emergent conventions in multi-robot systems. The robots are supplied with reactive control rules that hopeful lead to cooperative group behavior [Arkin, 1992]. Shoham and Tennenholtz [1995] propose a set of social laws for a group of idealized mobile robots that allow them to move on a grid without colliding and interfering. The social laws are basically traffic has that control how the robots move. However, designing all necessary conventions is difficult and perhaps intractable.

- **Learning** is an automatic method that incitement multi-robot to cooperate. The learning acquires the conventions (policies) by learning. This policies map a description of the environmental state to a set of robots behaviors. Matarić [1994b; 1995b; 1995c; 1997a; 1997c] used behavioral-based approach and she focuses on local interaction between robots and their environment as in reactive control. She attempts to increase both the environmental and cognitive complexity of behavior systems by first allowing multiple robots to act on the world, and second allowing robots to learn their complex group behaviors. She develops basic behaviors for spatial domain as building blocks for more complex group behaviors. This approach is useful for non-time-critical applications involve numerous repetitions of the same activity over a relatively large area. Parker [1994; 1995b; 2000b] extended her system to incorporate learning in a system called L-ALLIANCE architecture for learning intentionally cooperative among team of physically heterogeneous robots. This done by adjusting the parameters of motivational behavior that is responsible of guided the robots’ cooperation.

### 3.13 Reinforcement Learning in Multi-Robot Systems

Reinforcement learning has been used for learning control policies of behavior-based robots. Mahadevan and Connell [1992] have applied delayed Q-learning in a slightly different manner to learn the component behaviors within a predefined sequence, in which sub-goals were
introduced to provide more immediate reward. The particular task they investigate is for a robot to find, then push a box across a room. The task is divided into three behaviors: find-box, push-box, and unwedge-box respectively. Each sub-task is assigned to a module consisting of a reinforcement function and an applicability condition. The reinforcement function determines how the behavior should act given the correct sensory information. This function is learned using reinforcement learning, with a separate reward for each behavior. The applicability condition determines whether the action output by the module’s reinforcement function should be executed or not and conditions under which the robot transition from one behavior to another. The applicability conditions are not aware of other modules and can thus not prevent several modules is in control at any given time. In this application, the state space is huge, Mahadevan and Connell sought ways to reduce it, including weight Hamming distance and statistical clustering to group similar states. Using this approach, their robot OBELIX was able to learn to perform better than hand-coded behaviors for box-pushing. The sequence of behaviors is similar to the temporal-sequence approach (cf. §2.5) and learning at these elementary behaviors is equivalent to learning the state transitions of an FSA. An important difference is that learning takes place in behavioral states. The significant of this result is that Q-learning is useful in learning sequences within sequences of behaviors; it may be applied at several levels. Lin [1992; 1993] developed a method for Q-learning to be applied hierarchically, so that complex tasks are learned at several levels. He argues that by decomposing the task into sub-tasks and learning at the sub-task and task level, the overall rate of learning is increased compared to monolithic learners. In Lin’s work, the task decomposition and assigning reward functions and application space to sub-tasks job is carried out by designers, the robot learns the reset. The application space is not defined by a subset of the possible inputs, but is rather a subset of the state space as a whole. Lin’s results show significantly faster convergence and better performance in comparison with monolithic technique. However, Lin concentrates on tasks where subtasks combine to solve a global task, but one may equally apply the architecture to tasks where the sub-tasks where the sub-tasks are overlapped and interfered with each other. Similarities between Lin’s decomposition and temporal-sequencing for assemblages of motor schemas (cf. §2.5) are readily apparent. Lin’s sub-tasks or elementary skills correspond to behavioral assemblages, while a high-level skill is a sequence of assemblages. Learning at the higher-level is equivalent to learning the state-transitions of an FSA and learning the elementary skills corresponds to tuning individual states or behavioral assemblages. Humphrey [1995; 1997] presents a behavior-based
algorithm where the behaviors of a robot are considered agents competing for control of the robot. The agents limit their inputs and each performs Q-learning with local reinforcement functions. The algorithm known as W-learning where each agent suggested an action with some weight based on its Q-values and the robot executes the action with the highest weight. Several arbitration strategies are attempted, varying which measure to use (e.g. Maximize happiness or minimize unhappiness) to determine a winner in a competition for control of the agent. The system manages to learn an approximate solution, though no mention is made of how the learning time compares with that of a monolithic agent. Furthermore, by only considering competitive arbitration strategies, the approach may be limited in what types of goals it can handle. Matarić [1994b; 1994c; 1995a; 1997c] proposed a reformulation of Q-learning reinforcement learning paradigm in a group of robots called Nerd Herd. She was using higher levels of abstraction (conditions, behaviors, and heterogeneous reward functions and progress estimators instead of states, actions, and reinforcement) to enable robots to learn a composite foraging behavior. The heuristic reinforcement function is based on principled embedding of domains knowledge. The foraging behavior consists of a workspace of multiple robots gathering pucks and transporting them to a home location. At any given time, the robot can chose between three behaviors, avoid, grasp-puck, and drop-puck, which are triggered automatically under proper condition. The first class rewards the pre-coded behaviors, giving positive rewards when a puck is grasped or dropped in the home location and negative reward if a puck is dropped in some location other than home. The progress measurements are triggered by an external event (detecting another robot or grasping a puck), and give positive feedback if the robot moves in the right direction (away from the other robots or towards home) and negative feedback otherwise. After a set period of time, the robot forced to switch behaviors and monitoring is turned off. The heuristic approach is shown to perform significantly better than Q-learning and Matarić concludes that Q-learning is not appropriate for multi-agent learning tasks. The presented benefit of this work is the speed of learning achieved by using progress monitoring reward functions [Matarić, 1994c]. While this work illustrates the benefit of using prior domain knowledge encoded as reward functions, simply using reinforcement learning to learn how to arbitrate between behaviors is probably not feasible in most complex domains. Simsarian and Matarić [1995] using the Q-learning to multi-robot box pushing. They implement their algorithm on the two six legged mobile robots. Balch [1998a] integrates the reinforcement learning and schema-motor control for multi-robot system to study the impact of behavior diversity on the cooperation.
among multi-robot systems [Balch 1999; 2000]. He discovers that there is a tied relation between the diversity and mission done multi-robot. In other words, the mission in hand affects the behavior diversity in hand. He concludes that the uses of global rewards lead to great diversity and that global rewards are the better choice in the domains where behavioral diversity is important [Balch 1998b]. This because it is more likely robots will be rewarded for different behaviors under global reinforcement. Similarly, local rewards are the best choice in tasks where homogenous behavior is preferred. Touzet [2000] has combined the instance-based or (lazy) learning and the Q-learning together in what called lazy Q-learning. The lazy learning provides an instantaneous set of situation-action pairs represent the situation transition function. Lazy learning samples the situation-action space according to a random action selection policy, storing the succession of event in memory, and when needed, probes the memory for the best action. Probing is done using a modified version of the technique proposed in [Sheppard 1997] by predicting the rewards for state-action pairs without explicitly generating them. For the current real-world situation, a situation matcher locates all the states in the memory that are within a given distance. If the situation matcher has failed to find any nearby situations, the action comparator selects an action at random. Otherwise, the action comparator examines the expected rewards associated with each of theses situations and selects the action with the highest expected reward. This action is then executed, resulting in a new situation. On the other hand, Maes and Brooks [1991] present a distributed statistical learning reinforcement algorithm that learns to coordinate the behaviors necessary for six-legged walking in an insect-like robot called Genghis. Each of the robot’s six built-in swing-legged-forward behaviors can be considered as an agent. Their architecture contains a set of perceptual conditions, a set of behaviors, a positive feedback generator, and a negative feedback generator. The behaviors all learn in parallel, receiving immediate global reinforcement signals. In their scheme, each behavior learns itself when it ought to be applied. They predefined a set of behaviors and a set of binary perceptual conditions. Each behavior learns when it should be “on” or “off” based on perceptual conditions. Positive and negative feedback are provided to guide the learning. The behaviors learn, for each perceptual condition, relevance and reliability of the behavior to the condition. A behavior is relevant in the presence of a particular condition if it is positively correlated to positive feedback (i.e., positive feedback is likely to be received if the behavior is activated in that condition). The correlation for positive feedback for a behavior B is computed by:
\[ \text{corr}(P, B) = \frac{(j \times m) - (l \times k)}{\sqrt{(m+l)(m+k)(j+k)(j+l)}} \]  

(3.19)

Where \( j, k, l, \) and \( m \) are the behavior’s performance measures for the number of times that the feedback is “on” or “off” and whether or not the behavior is active or not during that time as shown in Table 3.6.

<table>
<thead>
<tr>
<th></th>
<th>Active</th>
<th>Not active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive feedback</td>
<td>( j )</td>
<td>( K )</td>
</tr>
<tr>
<td>No positive feedback</td>
<td>( K )</td>
<td>( m )</td>
</tr>
</tbody>
</table>

**Table 3.6: The Statistics Parameters Meanings.**

The same equation is used for correlating negative feedback and the behavior’ relevance is determined by computing the difference between positive and negative correlation as follow:

\[ \text{corr}(P, B) - \text{corr}(N, B) \]  

(3.20)

A behavior is reliable if the probability of receiving the feedback is close to 1 calculated from the next equation.

\[ \min\left(\frac{j_p}{j_p + l_p}, \frac{l_p}{j_p + l_p}, \frac{j_n}{j_n + l_n}, \frac{l_n}{j_n + l_n}\right) \]  

(3.21)

The behaviors learn both negative relevance (when they should be turned off) and positive relevance for each condition. Conditions that are neither positively or negatively relevant are eventually dropped from consideration. The above method works will because the reinforcement signal is immediate and the individual behaviors do not need to be learned. It is also ideally suited for the task of coordinating multiple actuators. Parker [1992; 1994; 1995a; 1996; 1998a] developed the ALLIANCE architecture for controlling teams of physically heterogeneous robots. The system is built on the behavior-based subsumption architecture, where the tasks are broken into sub-tasks, with groups of behaviors addressing each sub-task. At the highest level, mutually inhibitory *motivational* behaviors direct the overall behavior of the robot, activating in turn lower-level behaviors that combine to solve the sub-task. Along with the typical sensor-based conditions that might trigger motivational behaviors, Parker adds *impatience* and *acquiescence* as two mathematically modeled motivations.
to achieve adaptive action selection. Impatience increases if no other robot is attempting to solve the sub-task associated with a motivational behavior, while acquiescence inhabits a behavior if the robot is not meeting with success. The combined result of the ordinary conditions, impatience, and acquiescence in a group is that the group cooperates in striving to solve the overall task. Where the execution of a behavior one robot directly inhabit the execution of the same behavior on another. Parker [1994; 1995b; 2000b] extended her architecture by L-ALLIANCE by incorporating the use of performance monitors for each motivational behavior within each robot as shown in Figure 3.3. Each monitor behavior is responsible for observing, evaluating, and cataloging the performance of any robot team member (including itself) whenever it performing the task corresponding to that monitor's receptive behavior set. The L-ALLIANCE in its two-control phase of learning implements a statistical reinforcement learning for control parameters adaptation for adjusting activation thresholds used to derive the action selection strategy in the multi-robot system domains. In its active learning phase, the robot motivational behavior interact to cause each robot to select its next action randomly from those actions that are: (1) currently undone, as determined from the sensory feedback, and (2) currently not being executed by any other robot, as determined from broadcast communication messages. While they perform their tasks, the robots are maximally patient and minimally acquiescent.

Figure 3.3: The L-ALLIANCE architecture (adopted from Parker, 1994).
However, during the *adaptive learning phase*, robots have to make concentrated effort to accomplish the mission with whatever knowledge they may have. Thus, the robots acquiesce (give up tasks) and become impatient (take over tasks) according to their knowledge and the control strategies. Accordingly, the motivation of robot $r_i$ to perform behavior set $a_j$ at a time $t$ is calculated as follow:

$$m_j(0) = 0$$
$$m_j(t) = [m_j(t-1) + \text{random} \_ \text{increment}] \times \text{sensory} \_ \text{feedback}_j(t) \times \text{activity} \_ \text{suppression}_j(t) \times \text{learning} \_ \text{impatience}_j(t) \quad (3.22)$$

Equation (4.4) is used during the active learning phase. For adaptive learning phase, the Equation (4.5) is applied.

$$m_j(0) = 0$$
$$m_j(t) = [m_j(t-1) + \text{impatience}_j(t)] \times \text{sensory} \_ \text{feedback}_j(t) \times \text{activity} \_ \text{suppression}_j(t) \times \text{impatience} \_ \text{reset}_j(t) \times \text{acquiescence}_j(t) \times \text{learned} \_ \text{robot} \_ \text{influence}_j(t) \quad (3.23)$$

The L-ALLIANCE has been successfully used for a wide range of mission scenarios on both physical and simulated mobile robots.

### 3.14 Task-Allocation and Cooperation

Multi-robot cooperation is an instance of dynamic task allocation or role\(^6\) allocation in distributed manner, which enables the system robust to failure and assists in the achievement of group(s) goal rapidly and optimizes the global system behavior by reducing interference and conflicts that arise among robots [Ostergaard *et al.*, 2002]. Dynamic task allocation has been extensively studied in social insects, which is interesting since their labor division and its regulation are organized by surprisingly simple and robust means. One of the most inspiring models that concentrate on decentralized and flexible task allocation is the *activation-threshold* model, in which individual agents react to stimuli that are intrinsically bound to the task to be accomplished [Bonabeau *et al.*, 1999]. From behavior-based robotic perspective, a significant amount of previous research has been accomplished in the areas of cooperation of multi-robot systems and optimal task allocation/scheduling. For collective robotics cooperation,

---

\(^6\) The word *task* and *role* may be used one for the other.
Goldberg and Matarić [1997; 1999] have investigated task-specialization, or role–based task division in foraging systems, with and without explicit communication. They suggest pack and caste arbitration as a mechanism for generating efficient behavior and reducing interference. In pack arbitration, all of the individuals of the robot group are behaviorally homogenous capable of performing any of the behaviors necessary for the group to complete the mission. The arbitration is based on dominance hierarchy, which force the individual lower in dominance hierarchy to defer to the higher. Since, all of the robots are capable of the same behaviors, the loss of any one individual will not make achieving the goal impossible, unless no robot remain. This form of arbitration needs some information that must capable to communicate, either explicitly or via direct sensing is short-term goals, location, and position in the hierarchy. In caste arbitration, all of the individuals of the robot group are behaviorally heterogeneous and are not capable of accomplishing all of the tasks that the group requires for completing the mission. The arbitration is based on specialization, which force the individual to do the tasks, which are related to its capability. Ostergaard et al., [2001a] implement simple algorithm for emergent bucket brigading among foraging robots without the needs for imposing territorial bucket brigading as Fontán and Matarić [1996; 1998] by using territorial arbitration, each robot has its own territory, which no other robot may enter, by letting a group of forager robots that collect pucks within their specific mutually-exclusives, and deliver them to the boundary of the next territory closer to the home region. For intentionally cooperative robotics; Parker [1994; 1998a] used the idea of dynamic task allocation in a decentralized multi-robot architecture by implemented an activation-threshold approach by mapped the robot’s ability to perform a task to a scalar quantity, which is used to assign tasks to robots. Using this method the team of robots has to be heterogeneous and each robot has to be characterized by a different threshold in order to regulate the activity of the team. She described multi-robot experiments with a priori hard-wired heterogeneous capabilities using ALLIANCE architecture, which has a mechanism of impatience and acquiescence, promoting robustness and fault tolerance. She used motivational behaviors to store information about other individual robots. A toxic waste cleanup used in the work and she describes a temporal-division that sends one robot to survey and measure the environment, and then has the reset of the group use its information to clean up the spill. This approach is a form of caste arbitration with one output of one caste serving as inputs to another. The same work also describes a simulated office garbage-collection task in which the division of the labor is performed based on first-come first-serviced basis using a simple and elegant conflict-
resolution scheme. Parker [1994, 1995b] extended her architecture by L-ALLIANCE implemented a style of parameters learning for adjusting activation thresholds used to perform dynamic task-allocation and task-ordering in her multi-robot system (cf. §4.2). Werger and Matarić [2000a; 2000b] provide broadcast of local eligibility (BLE) architecture, a distributive derivatives of Subsumption Architecture [Brook, 1986], to coordination among multiple robots by having the execution of the behavior on one robot directly inhabit the execution of the same behavior on another. The cross-inhabitation of behaviors is an opportunistic approach for distributing the tasks. In the BLE framework, robots have no commitment to their tasks; the relevant behaviors continuously communicate to decide who is best fit for each task, so that the tasks must be mutually exclusive. Gerkey and Matarić [2000; 2001a; 2001b; 2002] presents “Murdoch”; a completely distributed, resource-centric, publish/subscribe communication model. The approach is based on commitment, where a task allocation strategy using a market-based auction system commits the robots their tasks until success or failure. Firstly, each robot bids on a task based on its perceived fitness to perform the task. Secondly, a single round auction decides which robot gets the task. Finally, the winning robot’s controller performs a sequence of actions to execute the task. A key feature of this approach is that all communication is resource-centric and never name-based. On the side, the instantaneous greedy task scheduling in Murdoch does not allow for opportunistic optimization. Ostergaard et al., [2001, 2002], based on biding mechanism, studied four different task allocation strategies, which derived from the combination of two variables: the amount of commitment to a given task engagement, and the amount of coordination among the robots.

3.15 Discussion and Summary

In this chapter, we have presenting the basic reinforcement-learning framework and then discuss some popular algorithms and techniques. Moreover, we discuss which algorithm is suitable for our tasks. We discuss also the integration of planning within the reinforcement algorithms and how the reinforcement functions are affected by the designer needs. Moreover, we present other reinforcement techniques and a comparison between them to show their pros and cons. Moreover, the chapter reviews the important existing work related

---

7 There are many important reinforcement-learning topics not discussed here like: the generalization techniques for continues and large discrete state spaces, which implemented by popular techniques such as neural networks and Cerebellar Model Articulation Controller (CMAC) (Albus, 1981), and hierarchical problem solving.
to the dissertation. It overviewed the cooperation methods used in behavior-based multi-
robot, and multi-robot learning works are reviewed. Finally, the work task allocation
discussed.

Cooperation is a form of interaction among robots. This interaction can be regulated and
amplified through a certain amount of extra-work and can be achieved by three methods,
those based on communication, those based on convention and those based on learning.
Communication and conventions (social rules) are static methods of cooperation lie in the
imposition by the designer of a control or communication structure in the interaction
dynamics of multi-robot behaviors. Though communication is often helpful and
indispensable as an aid to robots group activity, it does not guarantee cooperative behavior, is
time consuming, and can detract from other robot behavior if not carefully controlled or
coordinated. At other times, communication can be risky or even fatal (as in some hostilities
situations where adversary can intercept communication messages). Even for tasks in which
environmental information unavailable, more communication does not always equate to
more information. However, communicating autonomous robots have several desirable
features: they are locally organized, which is nice because it reduces the complexity of
designing a system, and lends them a robustness that hierarchical systems can not match.

Conventions (social rules) are static methods of cooperation lie in the imposition by the
designer of a control structure in the interaction dynamics of multi-robot behaviors. The
robots are supplied with reactive control rules that hopeful lead to cooperative group behavior.

In reactive and behavioral cooperative systems have a number of problems? First, because
the robots do not model each other or the world, their performance is often haphazard;
robots undertake so-called cooperative actions without any sense of whether are actually
helping the robots with whom they are supposed to be cooperating. In addition, the
emergent nature of reactive or behavioral intelligence puts the burden of deciding how to
react to various state of the environment on the programmer. Learning can be used as an
adaptive and an automatic method that incitement multi-robot to cooperate by acquiring the
conventions (social rules) through trial and error. On the other hand, by developing policies
between robots that complements each other. The research in this dissertation differs from other
work in several important respects. The work is distinguished by the fact that learning robots are
the central investigative mechanism. No responsibility is made in advance to any particular
robot or arbitration mechanism. This is will happen dynamically. This will open the

48
possibility that the robot may discover new form of dynamic task-allocation and cooperation themselves.
CHAPTER 4

LEARNING COOPERATION BY REWARDS DISTRIBUTION

There has been an upsurge of interest in multiple robots engaged in cooperative behavior in recent years due to their applicability to various tasks. However, without proper mechanism that guides robots' interactions, the mere result of gathering multiple robots is chaos. The resolution of conflicts in robots' behaviors and obtaining cooperation amongst them is a necessary but difficult problem. This chapter outlines the proposed research undertaken in studying appropriate conflict resolution rules among multiple robots engaged in collective behavior through cooperation using a new formulation of distributing the Q-learning algorithm. Distributed Q-learning projected a centralized policy for its components to yield distribution on policies for autonomous multi-robot system. The projection produces four schemes of distribution; by using a reward distribution scheme the objective is to enhance the performance, throughput, and cooperation of the team members.

4.1 Introduction

Purely distributed multi-robot systems, which do not have access to other robots' policies cause, what is known as the problem of non-stationary learning policies. The main caused of this problem is, from the respective of a single robot, as the other robots learn, they adjust their policies for choosing actions; hence the state transition probabilities change correspondingly. However, robots need to dynamically allocate responsibilities for different subtasks depending on the context of each robot as well as changing circumstances of the overall situation. There are two approaches for relating information to each robot about its team members in the reinforcement-learning framework to overcome the problem of non-stationarity. One approach is to augment the state vector of each robot to include this information, where a robot is “aware” of other robots’ perceptions and actions, using communication scheme. However, this approach is difficult to implement in noisy environments where robots cannot reliably discern states and actions of other robots and make the team as a “big robot.” In this case, the perception and action are the union of all team members’ perceptions and actions [Version and Gambardella, 1996].
<table>
<thead>
<tr>
<th>Problem</th>
<th>Learning from distributed rewards.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion</td>
<td>Centralized reinforcement learning must be distributed.</td>
</tr>
<tr>
<td>Approach</td>
<td>Project this centralized policy for its components to yield distribution on policies. Using distribution of weighted-sum of future rewards.</td>
</tr>
<tr>
<td>Validation</td>
<td>Implementing learning on a group of mobile robots learning to forage.</td>
</tr>
</tbody>
</table>

Table 4.1: A summary of the situated learning problem addressed here, and the structure of the proposed solution.

Another approach is to share information gathered from reinforcement learning algorithm of each robot to reflect the performance of other robot. Distributing this information among robots reduces the size of the search area, and hence the search time, by providing the learner(s) with auxiliary sources of bias. When a distributed expertise is not information, the search through the state space is unbiased and can be excessively long [El-Telbany et al., 2001, 2002a]. Tan [1993] has extended the use of reinforcement learning to multi-agent systems cooperation through policy sharing and exchanging policies. In policy sharing, each agent uses the same policy for selecting an action, although behaviors between these homogenous agents differ since they receive different inputs. The advantage of policy sharing is that training can be very fast, but the main disadvantage is that it does not allow for task-specialization or the development of self-interested agents. In addition, exchanging policies is much costly and requires huge real-time communication (cf. §4.7 for a comparison with our proposed approach). In contrast to the research work mentioned above, we present a novel and distributed framework to allow robots to get access to some information about the other robots. Using such information, robots can potentially perform much better than if they were to observe only the effect that other robots have on the common environment. In addition, robots can specialize their expertise to specific classes of tasks that can arise in the environment. This chapter will address the following problem: how can collections of homogenous robots learn cooperation by distributed reinforcements? We propose a formulation of reinforcement learning where an elegant distributive Q-learning algorithm for cooperative multi-robot systems is introduced. The distributed Q-learning algorithm based on the estimation of distributed weighted-sum of future rewards and the q-function values for the entire robot team. Thus, by treating a collection of robots as a single robot that uses a central optimal MDPs policy, and projecting this centralized policy for its components, a distributed policies for autonomous
multi-robot system will emerge. Demonstrating an effective learning algorithm on a group of robots learning to forage validates these ideas. Table 4.1 summarizes the problem and the approach.

### 4.2 Distributed Q-Learning

Considering a system that includes multiple robots, each has its own set of possible actions and its own observations. The team members must coordinate in order to achieve a common goal. Extension of reinforcement learning algorithm to the coordination of multi-robots case is implemented by treating the collection of robots as a single robot using a central controller. Therefore, the finite set of all possible actions \( A = \bigcup_{i=1}^{n} a_i \) is the sets of joint actions for \( n \) robots. This is modeled as a single robot with a finite set of global states \( S \), a transition rule \( T(S_t, A_t, S_{t+1}) \) and a global reward function \( R: S \times A \mapsto \mathbb{R} \). The aim of learning is to find an optimal policy \( \Pi: S \mapsto A \) that maximizes the discounted sum of future rewards accumulated by the multi-robots as follows:

\[
\sum_{t=0}^{\infty} \gamma^t R(S_t, \Pi(S_t))
\]

The exact Q-values for all state-action pairs can be found by solving the Bellman equation (cf. Equation (3.8)), which have the following matrix form:

\[
Q^\Pi = R + \gamma T^\Pi Q^\Pi.
\]

Where \( Q^\Pi \) and \( R \) are vectors of size \( |S| \times |A| \), and \( T^\Pi \) is a stochastic matrix of size \( (|S| \times |A|) \). Since, we are interested in the distributed learning of complete independent policies, to form an optimal behavior by multi-robots, each robot plays the role of the learner and trainee must autonomously perform its own actions, obtaining its reward and update its q-values regardless to the actions performed by other robots. This it can be done by splitting the global optimal policy \( \Pi^* \) into \( n \) component-policies \( \pi_i: s \mapsto a \) with \( \Pi^*(S) = (\pi_1(s), \cdots, \pi_n(s)) \). So constructing multiple q-functions depend only on the robot its local state and its action only. A weighted function is required to distribute the
information of the centralized $Q$-function to the smaller individuals $q$-functions. The $q$-functions are computed as follows:

$$q^* = \mathbf{P}^* \cdot (R + \gamma T^\Pi Q^\Pi)$$

(4.3)

Where $\mathbf{P}^*$ denotes the probability of state and action sensed and chosen by a robot appearing in global state and action for the global reward and $Q$-function, and $T(S, A, S')$ is the transition probability. In other words, $\mathbf{P}^*$ represents the \textit{weights of cooperation} among the robots in the distributed form by reducing their locality through communication.

**Figure 4.1: Cooperative and Distributed $Q$-Learning.**

Since every robot picks up its action from the large $Q$-function, only the actions, which are maximally taking into account cooperation with other robots, will act optimally. We can rewrite the previous equation in a distributed form as follows:

$$q^*_i(s, a) = \sum_j G_{ij} R_j(s, a) + \gamma \sum_j F_{ij} \sum_{s' \in S} T(s, a, s') \max_a q^*_j(s', a')$$

(4.4)

Where $G_{ij}$ and $F_{ij}$ are weighting functions that determine the level of contribution of distributed value in the overall system, as shown in Figure 4.1. Selection of weight values for both $G_{ij}$ and $F_{ij}$ leads to different forms of distribution in multi-robots reinforcement.
learning algorithms and consequently different forms of coordination. Within this formulation, the most known distributions schemes for reinforcement learning update rules are four, the first two are common in literature in various forms, while the last two are new. Following the classification of Schneider et al., [1999] we get four types of distribution:

1. **Global Reward Distributed Reinforcement Learning (GRDV):** Balch [1998] and Crites and Barto [1998] used this scheme, where the decision-making for each robot is locally but learning and control decision are based on a global reward function. This means that the weight $G_{ij}$ does not distribute the global reward function but uses the local state $q$-values with $F_{ij}$ is unity for local value function and zeros for other robots, and the resulted learning rule is:

$$q^*_i(s,a) = R(S,A) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \max_{a'} \tilde{q}_i(s',a')$$

In this formulation, each robot aim to maximizes the discounted sum of future global rewards. This method is not fully distributed, though, since it depends on the broadcast of a global reward signal, and the composite policy is no longer guaranteed to be optimal for the overall system.

2. **Local Distributed Reinforcement Learning (LRLV):** Matarić [1994] and Balch [1998] used this scheme, where the decision-making for each robot is locally to optimize its own rewards and does not communicate with its team members. That results in the following formulation:

$$q^*_i(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \max_{a'} \tilde{q}_i(s',a')$$

Rather than assuming a global reward function that is computed and broadcasted to all robots, a local reinforcement signal available with each robot is used. It is computed directly from a robot’s own local sensors and actions. In this formulation, each robot aim is to maximize the discounted sum of future local rewards. Here, the values of $G_{ij}$ and $F_{ij}$ are considered unity for the robot’s local reward and value function, and zero for the others. This method is fully distributed and does not communicate with other robots; however, the composite policy is no longer guaranteed to be optimal for the overall system.

3. **Distributed Value Function Distributed Reinforcement Learning (LRDV):** Schneider et al., [1999] implement this scheme to overcome the locality and increase the
level of cooperation and the guarantee that distributed policies leads to an optimal composite policy is by communicating the value function of a robot among team members. To ensure the policy’s optimality, rewards accumulated in \( q \)-function are used for learning as follows:

\[
q^\pi_i(s,a) = R(s,a) + \gamma \cdot \sum_j q^j_i \cdot \sum_{s' \in S} T(s,a,s') \max_{a'} q^j_i(s',a')
\]  (4.7)

With this formulation, a distributed representation of the value function over the robots team is created. It is possible to sum the value function over all distributed robots and the result would be an expected discounted weighted-sum of future rewards of its team local rewards. This is accomplished through communication between robots teammates. Here, the values of \( G_{ij} \) are considered unity of the robot’s local reward and zero for the others.

4. Distributed Reward Distributed Reinforcement Learning (DRLV): El-Telbany et al. [2001; 2002] proposed and implemented this scheme to overcome the learning from value functions, which is not relaxed in the early stage of learning. Where the learning from distributed rewards is used and these rewards represent the true reflection of success. The resulted equation is:

\[
q^\pi_i(s,a) = \sum_j G_{ij} \cdot R_j(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \max_{a'} q^j_i(s',a')
\]  (4.8)

The weighting function \( G_{ij} \) determines how strongly each robot will weight the immediate rewards of other team members in addition to its rewards. This might lead to less greedy behavior among the distributed robots because everyone cares about himself and his team members. In this case, each robot’s aim is to maximize the expected discounted weighted-sum of future rewards of team local rewards as follow:

\[
\sum_{s=0}^\infty \gamma^s \sum_j G_{ij} \cdot R_j(s_i, a_i)
\]  (4.9)

The values of \( F_{ij} \) are considered unity for the robot’s value function and zero for the others. This is similarity to the previous formulation, except that the weights value for both \( G_{ij} \) and \( F_{ij} \) are not equal. Since \( q_i(s',a_i) \) is the estimate of the total discounted reward expected in
the future, both $G_{ij}$ and $F_{ij}$ are the parameters of distributed rewards. In other words, the weights values for $G_{ij}$ and $F_{ij}$ are not equal, and adapting the weights values for $G_{ij}$ or $F_{ij}$ is sufficient to get an optimized level of cooperation. Putting Equation (4.4) in the $Q$-learning rule form, leads to:

$$q_i(s_t, a_t) = q_i(s_t, a_t) + \alpha \left[ \sum_j G_{ij} \cdot R_j(s_t, a_t) + \gamma \sum_j F_{ij} \cdot \max_{a'} q_j(s_{t+1}, a') - q_i(s_t, a_t) \right] \tag{4.10}$$

In this thesis, Distributed Reward Distributed Reinforcement Learning (DRLV), the fourth learning rule will be investigated in detail as a new form for learning to allow robots to get access to some information about the other robots. Using such information, robots can potentially perform much better than if they were to observe only the effect that other robots have on the common environment. In addition, robots can specialize their expertise to specific classes of tasks that can arise in the environment. In comparison with other schemes: Global Reward Distributed Reinforcement Learning (GRLV), Local Distributed Reinforcement Learning (LRLV), and Distributed Value Function Distributed Reinforcement Learning, the proposed approach have the following advantages:

- Provides auxiliary source of bias.
- Learning from stable reward function.
- Adaptive estimation of weights function.

### 4.3 Distributed Rewards Weighting Functions

Finding weighting parameters $G_{ij}$ are a problem. The main objective for each robot is maximize its cooperative reward $r_{coop}$ as follow:

$$r_{coop} = \sum_j G_{ij} \cdot R_j(s_t, a_t) = G_{i1} \cdot R_1(s_t, a_t) + G_{i2} \cdot R_2(s_t, a_t) + \cdots + G_{in} \cdot R_n(s_t, a_t) \tag{4.11}$$

Where $n$ is the number of the robots in the team. Realizing the cooperative behavior among homogenous robot teams; the robot profit is adaptively calculated to consider tradeoffs efficiently between acting selfishly and sacrificing for some individual benefit in favor of the overall team benefit. Since selfish and altruistic behaviors usually are opposing, combining both
self-profit and other-profit in a fixed form seems to be questionable, and some kind of adaptive and dynamic combination of both criteria might help to increase the cooperative level in multi-robot domains. One of the suggested weighting parameters \( G_{ij} \) is \( \frac{1}{n} \) of team total rewards. So, cooperative reward is the simple average of team’ rewards.

\[
\begin{align*}
    r_{coop} &= \frac{\sum r_a}{n} \\
    \quad \quad \quad (4.12)
\end{align*}
\]

Where \( \sum r_a \) is the summation of the rewards \( r_a \) of \( n \) the robot’ team. This equation implicitly motivates for cooperation behavior, but explicitly not since the robots cannot specialize their expertise to specific classes of tasks that can arise in the environment.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Learning Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learning Technique</strong></td>
<td>GRLV</td>
</tr>
<tr>
<td><strong>Optimization Criteria</strong></td>
<td>Maximize the discounted sum of future global rewards</td>
</tr>
<tr>
<td><strong>Distribution Term</strong></td>
<td>Global reward</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>Elevators</td>
</tr>
<tr>
<td></td>
<td>Control,</td>
</tr>
<tr>
<td></td>
<td>Soccer</td>
</tr>
<tr>
<td></td>
<td>Foraging</td>
</tr>
<tr>
<td></td>
<td>Motion Formation, Box Pushing</td>
</tr>
<tr>
<td></td>
<td>Power control</td>
</tr>
<tr>
<td></td>
<td>Foraging</td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<td>No Comm.</td>
</tr>
<tr>
<td></td>
<td>Min. search space/time.</td>
</tr>
<tr>
<td></td>
<td>Provides auxiliary sources of bias.</td>
</tr>
<tr>
<td></td>
<td>Provides class of learning rule.</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Cost of comm., Provide specific learning function.</td>
</tr>
<tr>
<td></td>
<td>Provide Specific learning rule Greedy policy, Non-stationary policy.</td>
</tr>
<tr>
<td></td>
<td>Learning from non-relaxed value functions, Cost of comm., Estimation cost of weights.</td>
</tr>
<tr>
<td></td>
<td>Cost of comm., Estimation cost of weights</td>
</tr>
</tbody>
</table>

Table 4.2: Summary of four learning schemes.
There are two forms for evaluating the cooperative rewards $r_{coop}$. One is a convex combination of the robot selfish weights applied to its selfish reward, and cooperative reward for each robot.

$$r_{coop} = w \cdot r_s + (1 - w) \frac{\sum_{a=1}^{n} r_a}{n}$$  \hspace{1cm} (4.13)

Where $w$ is the robot’s reward weighting, $r_s$ is the robot’s selfish reward, and $\sum_{a=1}^{n} r_a$ is the summation of the rewards $r_a$ of $n$ other robots in the team, respectively. The value of the parameter $w$ is obtained through learning, and determines the desire level to be selfish, and the desire level to be altruistic. This parameter ranges $(0 \leq w \leq 1)$, if $w$ is close to 1, the robot learns according to its rational behavior. With $w$ close to 0, learning from other robots is favored almost regardless of its own reward. The value of $w$ can be determined as static value or determined dynamically to reflect the “desire to be selfish”, and the “desire to be altruistic” based on the expected knowledge gain and previous experience.

The second form for evaluating the cooperative rewards $r_{coop}$ is:

$$r_{coop} = \frac{w \cdot r_s + \sum_{a=1}^{n} r_a}{w + n}$$  \hspace{1cm} (4.14)

The value $w$ ($1 \leq w \leq 3$) adjusts the selfishness gain of the robot. The range of $w$ was determined empirically through experimental results. If $w$ is close to 3, the robot learns according to its rational behavior, with $w$ close to 1, learning from other robots is favored almost regardless to its own reward. The robot, as well as receiving rewards from its own actions, receives a portion of the reward given by team members. Thus, each agent acts in a social way. Equation (4.14) is used through all the experiments presented and discussed in this research work.

### 4.4 Learning Adaptive Distribution of Rewards

The problem is how to learn adaptively the weighting parameters required to implement cooperation between multiple robots. Learning optimum values for *weighting-sum* parameters $G_{ij}$ determine the level of contribution of distributed rewards in the over all system. It is required to find an optimal criterion for weights determination that insures that the system may in *equilibrium*. The native way to find the value of $w$ is the using of optimization
technique. The optimization criterion is to maximizing the accumulated value stated in any of Equation 4.13 or 4.14. This is done with respect to event of activation for the optimization technique, which is occurring every time the local robot receives a positive reward (i.e., success). Using simulated annealing as our optimization technique, at each time step the robot receives a positive reward, that is broadcast to the con-specifics and adapted its reward weight \( w \) based on its rewards history from other robots’ team using simulated annealing. Adapting the weight value \( w \) takes two forms: (1) static and (2) dynamic adaptation. In static adaptation, estimation of the weight value \( w \) progressively increases with learning and continues until scheduling temperature reaches zero. Table 4.3, shows the complete algorithm for integrated simulated annealing learning and reinforcement leaning in multi-robot domains for static weights estimation.

| For each pair \((s \in S, a \in A)\), initiate the table entry \( q(s, a) \) to 0. |
| Observe the current state \( s \). |
| Set selfish weight \( w \) to 1.0. |
| Initialize scheduling temperature \( T = 0.2 \). |
| Set history list to empty. |
| Do forever |
| - Select an action \( a \) and execute it. |
| - Receive reward \( r \). |
| - Receive the con-specifics positive rewards \( r_c \). |
| - Save \( r, \sum_{a=1}^{n} r_a \) and \( n \) into history. |
| - If a positive reward \( r \) is obtained |
| - Broadcast it to the con-specifics, |
| - Tune the distribution weight \( w \) using simulated annealing based upon the history list. |
| - Get cooperative reward \( r_{coop} \) according to Equation (4.14). |
| - Observe the next state \( s' \). |
| - Update the table entry for \( q(s, a) \) according to Equation (3.16). |
| - Set \( s \) to \( s' \). |
| End Do |

**Table 4.3: Static linear estimation of the selfishness gain \( w \) using simulated annealing.**

In dynamic adaptation, the same objective is followed but with minor change. At each episode when the robot receives a positive reward, the selfish gain \( w \) is initialized to 0.5 and the temperature scheduling is reduced until it reaches zero to find the best value as shown in Table 4.4. This formulation is important since the environment of multi-robot foraging task is dynamic.
4.5 Experiments Result

In this section, some experimental results are provided in simulation to demonstrate the effectiveness of learning from distributed rewards and for static and dynamic weighting function parameters estimation using suggested algorithms listed in Tables 4.3 and 4.4. Both algorithms are evaluated using simulated multi-robot task. Where the simulation plays a viable role in experimentations. Since it enables the exploration of a large experimental space than would otherwise be possible. The quantitative performance data were gathered to determine the utility of the distributed reward reinforcement-learning scheme in comparison with other schemes to investigate their impact on the cooperation behavior.

For each pair \((s, a)\), initiate the table entry \(q(s, a)\) to 0.
Observe the current state \(s\).
Set selfish weight \(w\) to 0.5.
Set history list to empty.
Do forever
- Select an action \(a\) and execute it.
- Receive reward \(r\).
- Receive the con-specifics positive rewards \(r_a\).
- Put \(r, \sum_{a=1}^{n} r_a\) and \(n\) into history.
- If a positive reward \(r_s\) is obtained.
  - Broadcast it to the con-specifics,
  - Initialize scheduling temperature \(T = 0.2\)
  - Tune the distribution weight \(w\) using simulated annealing based upon history list until scheduling temperature reaches zero.
- Get cooperative reward \(r_{coo}\) according to Equation (4.14).
- Observe the next state \(s'\).
- Update the table entry for \(q(s, a)\) according to Equation (3.16).
- Set \(s\) to \(s'\).
End Do

Table 4.4: Dynamic estimation of the selfishness gain \(w\) using simulated annealing.

4.5.1 Domain of Experiments

In the experiments described below, the foraging task has been employed. It is widely used in the mobile robotics community. The foraging task involves the collection of objects of interest (attractors) scattered about the environment. In a typical strategy, a robot begins by wandering about the environment looking for attractors. Upon encountering an attractor the
robot moves towards it and grasp it. Then, the robot returns the object to the home base. Foraging has a strong biological basis. Many ant species, for instance, perform the forage task as they gather food. Foraging may find potential use in mining operations, explosive ordnance and waste or specimen collection in hazardous environments [Balch 1998a]. The performance in the foraging task is measured as the number of attractors collected and properly delivered by the robots in a time interval. Appendix B, details the simulator used and the used and the behaviors of the foraging task.

4.5.2 Performance Metrics of Learning Strategies

The evaluation of any system is often based on some metrics. The standard metrics for evaluating reinforcement-learning schemes for the multi-robot foraging task are [Balch, 1998a]:

1. Learning-rate, which is the time needed by the team to converge to stable behavior. This is measured by the number of times a robot policy changes during each trial.

2. Throughput, which was measured as the average number of attractors foraged, for different initial conditions experiments.

4.5.3 Experiments Setup

To measure the performance of the proposed algorithms, an experimental setup has been implemented. Data from twenty trials for each reinforcement strategy was collected and averaged on different number of robots, which ranging. Robot team involved up to seven robots and test in every experiment. In each simulation run, robots are allowed to learn until the total attractors collected, or an episode of 250 cycles of 10 seconds elapsed. The policy is represented in tabular form where there are eight states and three behaviors. The learning parameters for the experiments were set at $\alpha = 0.15$, $\gamma = 0.9$, $\epsilon = 0.25$, and $r = \pm 1$ (positive when an attractor is dropped at home, negative otherwise). Many factors determine the effectiveness of a cooperative multi-robot solution to a forage problem. Three main factors include the number of robots used, the physical distribution of the attractors and the obstacle density. All simulations consisted of a $10m \times 10m = (100m^2)$ world, with 40 attractors, 12 of them are stationary and the rest are mobile. In addition, the field includes nine obstacles varying from about $0.5m^2$ to $1m^2$ covering approximately 5% of the field. Each robot learns
to select the best behavior for each state, in order to find and take home the largest number of attractors, (cf. Figure B.1 in Appendix B).

### 4.6 Experiments

#### 4.6.1 Case I: Human Tuning for Weighting Parameters

This first experiment is designed to determine the effect of learning from distributed rewards on the multi-robot task performance. We have calculated the cooperative reward according to Equation 4.14, by manually changing the value of weight \( w \) in the range of 1.0 to 7. Figure 4.2 plots the average collected attractors (throughput) against the weight value for two, five and seven robots’ team in the world.

![The effect of weigh value on team’ performance](image)

*Figure 4.2: Average number of collected attractors elements for various numbers of robots with different reward weighing value.*

It is seen from the Figures (4.2), that when the weight value is high, the throughput was similar to that the selfish robot. However, for a weight value between 2 to 3, the performance is enhanced for the different number of robots. The main reason for this improvement is due to the solving of inter-robot credit-assignment problem. *Whence, this reward distribution reduces the conflicts and interference among robots by increasing the cooperation level* [El-Telbany et al., 2001; 2002a].
Figure 4.3 plots the average throughput of three learning strategies namely, locally rewarded, globally rewarded and distributed rewards reinforcement learning with best weight value tuned by human in comparison with the hand coded strategy for the number of robots in the team range from 1 to 7. It seen that the distributed reward increase the throughput of distributed robots and increase the level of cooperation among them as explained above. In fact we showed that $K$ robots learning cooperatively by distributed rewards over $N$ steps significantly outperform $K$ autonomous robots learning over $KN$ time steps. The lower performance of global reward due to in many situations confuse robots that they behave correctly or wrongly, when they are not actively involved in the activity leading to the reward.

Monitoring the number of changes occurs during learning investigate the level of volatility of the $Q$-learner under distributed rewards and local rewards. If it spiking with a less number of changes it is indicates that the $Q$-learner is nearly convergence. In order to show the learning rate of the team to converge to stable behavior the average number of policy changes per trial for four robots with different learning strategy is plot in Figure 4.4. The three learning strategies are showing good convergence properties. Robots using distributed rewards...
converge at a rate may be equal the globally rewarded robots. The locally rewarded teams do not fully converge, but settle to be around 15 changes/trial. The learning-rate metric can also be evaluated by observing how the throughput of the team changes over time. This can be called time-to-convergence, which is the number of trials required for team to discover correct policy. As shown in Figure 4.5, the three reinforcement-learning strategies increase their performance at a rapid rate over the first 50 trials and reaches its stable levels at 150 trials. However, the performance with globally rewards levels off at around 16 attractors. The locally rewards levels off at around 22 attractors. The distributed rewards levels off at around 25 attractors.

![The convergence of three learning systems with four robots](image)

*Figure 4.4: Convergence of learning for four robots under local, global and manually distributing rewards; lower numbers indicate convergence to a stable team policy.*

### 4.6.2 Case II: Adaptive Tuning for Weighting Parameters

This second experiment is designed to determine the effect of static weights adaptation on the distribution of rewards performance. Calculating the cooperative reward according to Equation (4.14), and automatically tuning the value of weight \( w \) in the range of 1.0 to 6 according to the algorithm listed in Table 4.3. The best estimated weights value for the seven robots teams are shown in Table 4.5. As presented in previous section, the performance of the robot teams enhanced when the weights lay in the range of 2 and 3.
The adaptation of weights in comparison with manually tuning weights as shown in Figure 4.6, increase the throughput of multi-robot task and increase the level of cooperation among them. This enhancement due to the estimation of optimal weights that is different from robot to robot instead of a fixed weight value for all the robots. The effect of the static adaptation on the learning rate in comparison with manually fixed weight is shown in Figure 4.7. We find that the fluctuation in learning rate is reduced and both of them converge to same number of changes. The reduction in fluctuation explains why the average number of throughput increased. The third experiment is designed
to determine the effect of dynamic weights adaptation on the distribution of rewards performance and comparing the results with the results obtained from the previous experiment. By calculated the cooperative reward according to Equation (4.14), using the algorithm listed in Table 4.4, the temperature scheduling is starting by a high value and reduced until it reaches zero to find the best weight value. *The dynamic adaptation process is implemented on the smaller scale and repeated every event (i.e., positive reward).* But for static adaptation, the scale is larger and occurring once by started and finished once the scheduling temperature reach zero.

The **performance of learning from distributed rewards with different weight's tuning**

![Graph](image)

*Figure 4.6: The average number of attractors collected by robot teams in foraging task under manual weight equals three and automatically (static) adaptive weight for distributed rewards.*

As shown in Figure 4.8, the throughput of distributed rewards by dynamic weight adaptation is increased in comparison with the throughput obtained by static weight adaptation this can be explained that, *the dynamic adaptation usually search for optimal weight value for each learning trial, while static adaptation search for a single optimal weight value for the learning process which can be not suitable for each trial conditions.* The learning rate of both adaptation methods is shown in Figure 4.9; in dynamic weight adaptation have little numbers of spiking and is smoother than static weight adaptation and stabilized more quickly.
Figure 4.7: Convergence of learning for under manually weighted of distributed rewards and static weight adaptation for distributed rewards algorithm; lower numbers indicate convergence to a stable team policy.

Figure 4.8: The average number of attractors collected by robot teams in foraging task under static weight adaptation and dynamic weight adaptation for distributed rewards.
The number of trials needed for the team to reach the steady state average throughput, which represents the time-to-convergence for the different approaches are shown in Figure 4.10. Using adaptive weight estimation increase the learning speed in comparison to learning with local reward and fixed selfish gain in reward distribution scheme, however both of static and dynamic adaptation there throughput performance reaches its steady state around 85’s and 75’s trials respectively, so there is no great convergence speed. As a result, the disadvantage of adaptive reward distribution using simulated annealing is its high computational complexity per time trial, which is independent of the sequence length of task execution.

![The convergence for learning from distributed rewards under different tuning methods](image)

**Figure 4.9:** Convergence of learning for under static weighting of distributed rewards and dynamic weighting for distributed rewards algorithm; lower numbers indicate convergence to a stable team policy.

### 4.6.3 Case III: The Effect of Communication Range on Performance

The third experiment is designed to determine the effect of communication range on the performance of proposed algorithms. In the previous two experiments, the communication range of the robot cover the whole environment length (i.e., 10 meters). By reducing the communication range to the half and quarter (i.e., 5 and 2.5 meters), the performance of the multi-robot system is reduced as shown in Figure 4.11.
Figure 4.10: The number of trials needed for steady state throughput for different learning schemes.

Figure 4.11: The effect of communication range on the distributed reinforcement learning strategy performance.
The deceases in the multi-robot’ performance is due the reducing of the global view of the robots and cooperative information. As there is no communication the performance becomes as the local rewards performance.

4.7 Discussion and Conclusions

This work presents in this chapter introduce the idea of the effect of including information about other robots into reinforcement signal function enhance the coordination capability of multi-robot systems, and increasing the guarantee of convergence of distributed $Q$-learning algorithms in the presence of multiple robots. In distributed rewards reinforcement learning algorithms, the adaptive weighting functions take the advantage of learns from its own problem-solving experience and experiences of the *conspecifics* in adaptive way. In the formulated distribute rewards scheme, it is not sufficient for each robot to act selfishly in order to find global optimal policy in multi-robot systems. In this cooperative mechanism, including information about other robot in the form of distributed rewards help to reduce the non-stationary in learning policies and the level of interference among them. Thus, they would have an enhancement and accelerating effect on any learning domains. However, there are limitations in the distribution reinforcement approach due to the computation cost of weight estimation and we do not specify in the update rule what state or action they have been or taken. In comparison with other schemes: Global Reward Distributed Reinforcement Learning (GRLV), Local Distributed Reinforcement Learning (LRLV), and Distributed Value Function Distributed Reinforcement Learning, the proposed approach have the following advantages:

- Provides auxiliary source of bias.
- Learning from stable reward function.
- Adaptive estimation of weights function.
CHAPTER 5

INTEGRATING APPRENTICE AND REINFORCEMENT LEARNING IN MULTI-ROBOT DOMAINS

This chapter outlines the works undertaken in studying the social behaviors among multiple autonomous robots. When multiple reinforcement learning robots engaged in collective behavior the apprentice learning (AL) arises, where each robot plays the role of learner and the trainer simultaneously. The objective of the learner is to learn a good policy that can be applied to the task, the robot attempts to meet this goal by learning from its own problem-solving experiences and experiences of the conspecifics. The objective and the major issue are to enhance the social interaction of multi-robot system through sharing experiences. Moreover, to overcome the drawback in adaptive rewards distribution arises from ignoring in the update rule what state and action they have been or taken in learning process.

5.1 Introduction

The ability to learn is an essential component of intelligent behavior in human society. Individual humans do not need to learn every thing by discovering it from scratch for themselves. Instead, they learn from their peers and teachers by exchanging the knowledge and expertise that they are acquired. Learning from peers does not occur only in humans. It has also been found in some invertebrates, in many birds, aquatic mammals and of course primates [Matarić, 1994a]. For an autonomous intelligent robot, by sharing the experiences of other robots performing the same task (or possibly performing the similar tasks), the overall learning speed and robustness of the solution can be increased. This increase in performance is possible because every robot receives, and hence gains from, the experiences of all the other robots virtually immediately.

Apprentice learning arises in multiple robots engaged in collective behavior, where each robot plays the role of learner and the trainer simultaneously. The objective of the learner is to learn a good policy that can be applied to the task. The robot attempts to meet this goal by

---

8 In apprentice learning, the robot tries to mimic a training robot’s problem solving behavior, for each state and associated action in the trainer’s solution the learner infers that the action was appropriate to perform in that state, so, the apprentice learning is a supervised learning technique where the training examples are derived from the trainer’s solution. Thus, the apprentice learning depends heavily on the trainer’s expertise. This method has been employed successfully to learn complex tasks such as game playing (Samuel, 1963), and vehicle control (Pomerleau, 1993).
learning from its own problem-solving experience and experiences of the con-specifics, since learning an optimal policy needs to many experiences. The relationship between the learner and the trainer in homogenous multi-robot domains, as shown in Figure 5.1, is similar to that in apprentice learning because the learner can observe the experience tuples that the trainer supplies, but the trainer does not need to provide experience tuple to the learner every step of the task. In a sense of social learning, training experiences serve as a source of vicarious learning for others [Matarić, 1994a; 1994b].

![Figure 5.1: The relationship between robots, the robots play two roles as a trainer and learner.](image)

Apprentice learning has the appeal for three main reasons:

- **First**, learning an optimal policy needs too many experiences; an apprentice learning is the suitable approach for learning by using the others’ experiences in multi-robot domains.

- **Second**, apprentice leaning process helped the learner to find the interesting parts of search space that are appropriate for learning an effective policy quickly by narrowing the search space.

- **Third**, training experience form the trainer does not needs any coding for its experience.

An integrated method needs to design learning algorithm that update the policy according to two sources of experiences (i.e., the robot its own problem-solving experience and others’ robots experiences). One of the earliest uses of apprentice learning was in training an agent to play the game of checkers [Samuel, 1967]. The trainers are master checkers players, whose move choices from many games are recorded as book moves. In learning to play, the automated agent simulates the re-playing of the recorded games, examining each game state in turn and adjusting its policy according to the move chosen by the master play.
Problem  Integrating apprentice learning in reinforcement–learning multi-robot domains.

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Speeding up learning using apprentice learning and enhancing the performance of learning from adaptive distributed rewards.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>Distribute own successful experiences among the team members. Thus make the robot plays the role of learner and teacher, and specify in the update rule what state and action they have been or taken in learning process</td>
</tr>
<tr>
<td>Validation</td>
<td>Implementing learning on a group of mobile robots learning to forage.</td>
</tr>
</tbody>
</table>

**Table 5.1**: A summary of the situated learning problem addressed here, and the structure of the proposed solution.

Lin [1992; 1993] proposed the learning from lessons in Q-learning algorithm, where a human trainer performs the task occasionally, leading the agent through all the steps necessary to progress from a start state to a goal state. The sequence of an experience tuple is captured in a lesson. After a lesson has been recorded, the learning agent incorporates it into its experience by simulating the trainer’s performance repeatedly. Lin’s approach requires the human trainer to develop complete sequences of actions that are appropriate for teaching the learner. Clouse and Utgoff [1992] add a simple interface to a reinforcement approach to allow a human to interact with the automated learner. The human monitors the learner’s performance and provides an action to the learner whenever he desires. Clouse [1997] integrated the apprentice learning and Q-learning in the ASK-FOR-HELP (AFH) framework, the learning agent employs Q-learning as its learning algorithm, and in addition the learner has access to actions provided by a trainer agent. The learner is completely responsible for its interaction with the trainer, asking the trainer to provide actions. The trainer always will give actions when asked. The learner employs one of two strategies of ask for help. With the first strategy, the learner asks the trainer to provide action randomly throughout training. The second asking strategy relies on the Q-values of the learner’s current action choices to determine whether to ask for aids or not. The difference between the Lin’s works is that Lin requires the trainer to perform the entire task, whereas AFH adapts according to the individual actions from the trainer. In contrast to our work, the trainer aids the learner when he gets a success (i.e., positive reward), which is a stochastic. The trainers’ aid gives an indication about its performance progress and directs the adjustment of robots’ policies in
the globally most beneficial direction by narrowing the search space [El-Telbany et al. 2002c]. This chapter will address the following problem: how can collections of homogenous robots learn cooperation and speedup learning process by distributed expertise? Moreover, learning from both of expertise and distributed rewards. A formulation of reinforcement learning in which the multi-robot teams play the two roles of learner and teacher simultaneously is proposed. Demonstrating effective learning algorithms on a group of robots learning to forage validates these ideas. Table 5.1 summarizes the problem and the approach.

5.2 Learning by Sharing Successful Expertise

Reinforcement learning methods require only experiences (sample state transition and rewards) \((s, a, r, s')\) obtained by interacting with the environment. If a robot has an almost correct world model, it can produce hypothetical experience to learn from them, since the learning process for an optimal policy needs many experiences. From an AI point of view, the optimal policy might obtained if the system had carried out explicit planning, which can be thought as learning the possible future robot-environment interactions, from the con-specifics to determine long-range consequences. Planning is learning from these hypothetical experiences and useful when making errors are costly, robot has less real experiences from the interaction with the environment, and speeding up the learning process is urgently [Sutton, 1990a; 1990b; 1991; Clouse, 1997]. Observing others’ interactions in successful situations is learning from hypothetical experiences generated by other robots and using such hypothetical experience, robots can potentially planning by learning from these hypothetical experiences. This planning process helped the learner to find the interesting parts of search space that are appropriate for learning an effective policy quickly by using the others’ team distributed successful experiences which represent the hypothetical experiences that learn from. In addition, building a model of the dynamics of the environment and its rewards. The model integrates all observed experiences so that the loss of useful information is minimal. Modeling approaches have several advantages besides their efficient use of experience, as if they are useful for exploring the environment, since they allow the robot to know what has not yet been learned. The overall architecture for the integrated algorithm is shown in Figure 5.2. This architecture enables the robots in multi-robots domains to learn by sharing their experiences. The real experience represents the basic interaction between the robot and the environment, giving raise to a trajectory of its own experience. The arrow on the left of the
figure represents direct reinforcement learning operation on its own real experience to improve the value function and the policy. On the right are training experiences planning process. The training experience are received from other robots, which reflect their success, and planning learning is achieved by applying reinforcement methods to training experience just as if they had really happened by the robot itself. The same reinforcement learning method used both for learning from real experience and from training experience.

The model proposed a new algorithm for integrating apprentice learning and reinforcement leaning in multirobot domains is presented as shown in Table 5.2. The algorithm speeding up learning process by planning in multirobot domains, where the robots learn to plan thought using the training experiences broadcasted from the con-specifics. The team members distribute their successful experiences (i.e., obtaining a positive reward), which represent the training or hypothetical experiences in apprentice learning, among themselves team members to modify their policies.

5.3 Learning from Expertise and Rewards

In the previous chapter, the distributed reinforcement learning schemes face a problem of ignoring in their update rule what state and action they have been or taken, as the learning from experience formulation presented in this chapter. So, it does not give an expected enhancement in the performance. Integrating both of learning from experiences and adaptive
distributed rewards into one algorithm is new. In this algorithm the robots learn to plan thought using the *training experiences* and *cooperative reward*. The aims from the integrated algorithm is to (1) *speeding up learning* and (2) *enhancing the throughput performance*. These aims based on:

- The training experiences are a form of planning gives an indication about global performance most beneficial direction by *narrowing the search space*.
- The cooperation rewards include information about other robot help to reduce the *non-stationary* in learning policies and the level of *interference* among them.

The planning occurs by using the *training experiences* broadcasted from the *con-specifics*, and *cooperation* as a result of weighting its reward and other rewards in the same situation and action if any. The complete algorithm is presented as shown in Table 5.3.

<table>
<thead>
<tr>
<th>For each pair ((s, a)), initiate the table entry (q(s, a)) to 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observe the current state (s).</td>
</tr>
<tr>
<td>Do forever</td>
</tr>
<tr>
<td>- Select an action (a) and execute it.</td>
</tr>
<tr>
<td>- Receive reward (r).</td>
</tr>
<tr>
<td>- Observe the next state (s').</td>
</tr>
<tr>
<td>- Update the table entry using <em>real experiences</em> for (q(s, a)) as equation (3.16).</td>
</tr>
<tr>
<td>- Set (s) to (s').</td>
</tr>
<tr>
<td>- If you obtained a success reward then</td>
</tr>
<tr>
<td>- - Broadcast your positive real experience to other robots.</td>
</tr>
<tr>
<td>- - If received success training experiences from con-specifics then</td>
</tr>
<tr>
<td>- - - Update the table entry using <em>hypothetical experiences</em> for (q(s, a)) as Equation (3.16).</td>
</tr>
<tr>
<td>End Do</td>
</tr>
</tbody>
</table>

**Table 5.2: A novel algorithm for integrated apprentice learning with reinforcement learning in multi-robot domains.**

### 5.4 Experiments Results

In this section, experiments results are presented to demonstrate the effectiveness of learning from sharing experience algorithm listed in Tables 5.2 and learning from expertise and distributed rewards listed in Table 5.3. The algorithms are evaluated using simulated multi-robot task, forage task and the same parameters and conditions described in pervious chapter (cf. §4.5), which can be summarized as follows:

- The data collected from twenty trials for each reinforcement strategy.
- The number of robots ranges from one to seven.
• The simulation episode is 250 cycles of 10 seconds.
• The learning parameters for the experiments were set at $\alpha = 0.15$, $\gamma = 0.9$, $\epsilon = 0.25$, and $r = \pm 1$ (positive when an attractor is dropped at home, negative otherwise).
• Simulation arena consists of a $10m \times 10m - (100m^2)$, with 40 attractors, 12 of them are stationary and the rest are mobile. In addition, the arena includes nine obstacles varying from about $0.5m^2$ to $1m^2$ covering approximately 5% of the field.

For each pair $(s \in S, a \in A)$, initiate the table entry $q(s, a)$ to 0.
Observe the current state $s$.
Set selfish weight $w$ to 0.5.
Set history list to empty.
Do forever
  • Select an action $a$ and execute it.
  • Receive reward $r$.
  • Observe the next state $s'$.
  • Broadcast your real experience to other robots.
  • If you receive hypothetical experiences for equal the state-action pair of real experience then
    • Get cooperative reward $r_{coop}$ according to Equation (4.14)
    • Update the table entry using real experience and hypothetical experiences for $q(s, a)$ as equation (3.16).
  • Set $s$ to $s'$.
  • If you obtained any successful reward from your real experience then
    • Tune the distribution weight $w$ using simulated annealing based upon the history list.
End Do

Table 5.3: A novel algorithm for integrated social and distributed rewards learning in multi-robot domains.

5.4.1 Case I: Learning from Training Experiences

In this experiment, the effect of learning from training experiences where each robot learns from its experiences in addition to other training experiences from the robot's con-specifics is investigated. The performances of this algorithm in comparison with the learning from local reward and adaptive distributed rewards are compared; the average number of attractors collected by the team members using training experiences for proposed algorithm is enhanced as shown in Figure 5.3. The reasons behind the improvement indicate that apprentice learning aid reinforcement learning to find a good policy by offering the training experiences from the con-specifics. Moreover, the training experiences are considered to the
source to directing the adjustment of policies in the globally most beneficial direction by \textit{narrowing the search space}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{performance_of_learning_systems.png}
\caption{The average number of attractors collected by robots in foraging task under different learning strategy. The training experiences enhanced the throughput of the team in comparison with local reward and distributed rewards policies.}
\end{figure}

As shown in Figure 5.4, the \textit{learning-rate} for learning from the dynamic weighting distributed rewards and training experiences. Learning from training experiences converges faster than the dynamic weighting distributed rewards, this can be also evaluated by observing how much the throughput of the system changes over time. Plotting the throughput versus trial number as the \textit{time-to-convergence}, determines the number of trials required for team to discover correct policy. As shown in Figure 5.5, when the training experiences are used, the throughput reaches its steady state in 50 trials approximately with a higher level of throughput, while dynamic distributed rewards and global and local rewarded learning team reaches its steady state in 75, 100 and 120 trials approximately.
Figure 5.4: Convergence of learning for under automatic (dynamic) weight adaptation of distributed rewards and distributed experiences algorithm; lower numbers indicate convergence to a stable team policy.

Figure 5.5: The time-to-converge rate curves for the four robots under leaning from different strategy.
5.4.2 Case II: Learning from Training Experiences and Rewards

In this experiment, the effect of learning from training experiences and adaptive distributed rewards is investigated where each robot learns its modified experiences using cooperative rewards in addition to other training experiences from the robot's conspecifics. The robot uses the apprentice learning to modify the policy by the experiences tuples, which are different from its experience, and distributed rewards to modify its reward from similar experiences. The performance of this algorithm in comparison with the learning from adaptive distributed rewards and training experiences are compared; the average number of attractors collected by the team members using training experiences for proposed algorithm is enhanced as shown in Figure 5.6. The learning-rate curve is shown in Figure 5.7. The number of policy changes for both rewards and experiences distribution algorithm is smoother with minimum number of spiking and converge to and remains at zero after 200's trial. The most important reason behind the enhancement in the performance is due to overcoming the problem of ignoring in the distributed rewards update rule the state or action they have been or taken and the two source of bias (i.e., experiences and rewards) directed the search space and reduce the non-stationary in learning policies and the level of interference among them. 

![Graph showing the performance of different learning systems.](image)

*Figure 5.6: The throughput of robots in foraging task under different learning strategy.*
5.5 Evaluating Reinforcement Learning Strategies

In this section, the reinforcement learning strategies proposed in this chapter and previous chapter are evaluated. The idea is based on the use of the interference\textsuperscript{9} and conflicts among robots as an evaluating property for cooperation in multi-robot systems. Let us consider that, if performance of a single robot $P$ and obtained under learning for $N$ steps, the expected performance of maximum cooperation among the $K$ robot is $K \times P$ for $N$ steps. This is acceptable if and only if the robots work at deterministic environments and have a global view for their environment. However, most of real application environments are dynamic and the robots have a local view for their environments. The interference among the robot teams is measured by calculated the difference among the actual team performance and their theoretical performance baseline. As shown in Figure 5.8, the interference and conflicts increased as the number of robots increase and can be seen from the divergence between the ideal performance curve and other learning strategy performance curve. As the robots increases the interference increases. Accumulated the

\textsuperscript{9}The interference among multi-robot systems is divided into types: physical and non-physical. In physical interference, the robots are in competition for the space. But, in non-physical interference, the competition becomes on the shared radio bandwidth, crossed infrared or ultrasound sensors, and algorithmically by undoing other the work of another (Goldberg and Matarić, 1997).
interferences for different learning strategy is shown in Figure 5.9. The accumulated interference of the distributed experiences strategy drops off to half of the locally rewards learning strategy. Generally the interference value is decreased in proportion to the performance.

Using the speedup metric $S[i, j]$ suggested by Arkin [1998], which measure the performance of a team of $N$ robots relative to $N$ times the performance of the single robot. Formally, the speed up for a team of $i$ robots curing out $j$ task is:

$$S[i, j] = \frac{P[i, j]}{P[1, j]*i}$$

(5.1)

Where $P[i, j]$ is the performance’ measure. The speedup results can be categorized into sub-linear performance ($S[i, j] < 1$), where multiples of a single robot performs better than a team; super-linear performance ($S[i, j] > 1$), where a team perform better than a team; and linear performance ($S[i, j] = 1$), a break-even point where the overall performance is comparable. As shown in Figure (5.10), the speedup increases for the proposed learning algorithms.
Figure 5.9: The interference level under different learning strategy in comparison with theoretically ideal cooperative teams.

Figure 5.10: The speedup metric of different learning strategy.
5.6 Flexibility of Learning Strategies

The flexibility refers to the ability of the robot member of team to modify its behavior as the environment or robot team changes. Ideally, the cooperative team should be responsive to changes in individual robot performance as well as dynamic environment changes. The need for the teams to be responsive to changes in robot performance reflect the fact that the capabilities of robots change over time due to learning or due to environmental causes that may reduce or increase a robot’s success. The flexibility of proposed learning strategies of reinforcement learning on the robot team performance is studied by changing the team members during learning process. This includes the followings:

- The effect of adding non-learnable robots to the learned team on performance.
- The effect of adding non-learnable robots to the non-learned team on performance.

![Figure 5.11: The performance of an added robot to a team of learned and non-learned robots using locally rewarded learning strategy.](image)

Twenty experiments had executed for different learning strategy using the same parameters and conditions described in chapter five (cf. §4.5) for only 100 trials. Figures 5.11, 5.12, 5.13, and 5.14 show the effect of joining a non-learnable robot to a team on its learning rate and performance. When a non-learnable robot joining a learnable team it’s learning rate increased faster than joining a non-learnable team. These results are reasonably since the robot learns nearly from extra
truthful sources of experiences that help it to reduce the size of the search area and guided it to right direction into an optimal learnable police. Moreover it’s performance reaches the same level as the learnable robot, which is learned for 300 trials. But for joining non-learnable team required many trials over that 100 trial to reach the same level.

![Figure 5.12: The performance of an added robot to a team of learned and non-learned robots distributed rewards learning strategy.](image)

![Figure 5.13: The performance of an added robot to a team of learned and non-learned robots using distributed experiences learning strategy.](image)
5.7 Multi-Robot Control Strategies: A Comparison

Designing reinforcement learning control architecture for multi-robot systems determine how multiple robots can communicate and assist in coordinating robot behaviors and cooperation between groups of robots. In this section, a comparison between centralized and decentralized control strategies is presented. One can characterize the techniques along dimensions such as: the policy size used by this control strategies, the Complexity of the strategy, which define the number of allowable, polices, the specialization of the control policy. One can also discuss how each control strategy deals with the robustness and scalability problems. Finally, another quality on which the methods differ is the speed with which each achieves its objective. One can compare and contrast these techniques along these dimensions and notice their pros and cons.

5.7.1 Centralized Control

Controlling multi-robot system by centralized controller can be considered as a single robot system, since there is a single central group policy as shown in Figure 5.15 (a), which maps a description of the multiple robots environmental states to a set of robots actions by union of all team members’ perceptions and actions [Version and Gambardella, 1996].
Although such a central policy may guarantee finding optimal cooperative policies of all the robots, the number of states and the product action space of the super-robot is usually much too large to be stored and its policy size equals: $A^N S^N$ and its complexity equals: $A^{N-1} S^N$, where $A$ is the number of actions, $S$ is the number of states, and $N$ is the number of robots. Furthermore, action selection will in general cost a lot of time. Finally, the control system is not scalable and not robust: if it breaks down, the performance of all robots collapses [Wiering et al., 2000]. Simsaian and Matarić [1995] and Matarić [1997b] use centralized policies for two robots in a box-pushing task. The robots receive the sensory information of other robots by communication and take turns in selecting the actions for both of them.

In order to make the policy space much smaller, a policy sharing is used. Where each robot uses the same single robot policy as showed in Figure 5.15 (b), for selecting an action, although behaviors between these homogenous robots differ since they receive different inputs. The advantage of policy sharing is that training them can be very fast due to the reduction of the policy size to $AS$ but a disadvantage is that does not allow for task specialization or self-interested robots and consequently not applicable for problems requiring different skills of robots [Wiering et al., 2000]. The complexity is reduced to $A^S$, a huge reduction compared to the centralized single group policy if there are many robots. Policy sharing has been shown helpful for simulated soccer [Salustowicz et al., 1998] and predator prey problem [Tan, 1993].

In this study, the foraging problem used in the previous sections was implemented using the policy sharing strategy using the same parameters and initial conditions of pervious experiments. As shown in Figure 5.16, the performance of the shared policy is better than

---

*There is only one robot keeps a decision policy and the other robots update this policy by their rewards communication.*
the local rewards and global rewards, however this enhancement not a significant due to the lack of specialization in the robots’ policies. Also, it was notice that, by adding mechanisms for cooperation (i.e., distributing the rewards) for distributed polices the performance enhanced significantly due to the diversity of policies arising from variation of roles and different accumulation of experiences, which changes during execution (cf. §5.9). Moreover, the policy sharing provided a faster learning in comparison with other distributed learning strategies as shown in Figure 5.17. This due to the robots profit by combining their experiences so that each time step more experiences are generated for updating the single policy.

The performance of three reinforcement learning strategies in comparison with shared policy learning strategy

![Graph showing the performance of different learning strategies](image)

**Figure 5.16:** The average number of attractors collected by robot teams in foraging task using shared policy in comparison with other three learning strategies.

### 5.7.2 Decentralized Control

The centralized policy space size can be reduced to \(NAS\) by breaking it down into a number of local policies each of them represent a robot as shown in Figure 5.18. In this case, each robot has to learn its own policy and the whole system is evaluated by examining the group behavior. The complexity of this system is: \(M^4\), since we have \(N\) polices of size \(A^5\). The advantage of decentralized approach is that facilitates the task specialization or role specialization, moreover the scalability and robustness for failures. The disadvantage of using such uncoupled control is that it may be hard for the system to find optimal cooperative
behaviors, since robots usually have a very local view and robots' behavior are not directly coordinated [Wiering et. al., 2000]. This problem overcomes by reducing the local view of the robots using communication to coordinate their behaviors [Matarić; 1997b]. Another possibility is to let robots learn models of other robots' behavior so that robots themselves can reason about and overcome possible conflicts [Nagayuki et. al., 1999]. Finally, an important topic in which the value functions and reward functions intake of the robot and its team is distributed and optimized [Schnieder et. al., 1999; El-Telbany et. al., 2001].

The convergence of policy sharing strategy in comparison with other three strategies

![Graph](image)

**Figure 5.17: Convergence of learning for under shared policy in comparison with other three learning policies; lower numbers indicate convergence to a stable team policy.**

The comparison among the two control schemes along the dimensions mentioned in previous sections is summarized in Table 5.4.

![Diagram](image)

**Figure 5.18: Decentralized Control Policies.**
<table>
<thead>
<tr>
<th></th>
<th>Single group policy (^{11})</th>
<th>Shared policy</th>
<th>Distributed policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>(A^{N+N}N)</td>
<td>(A^N)</td>
<td>(N.A^N)</td>
</tr>
<tr>
<td>Policy size</td>
<td>(A^N.S^N)</td>
<td>(A.S)</td>
<td>(N.A.S)</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Robustness</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Specialized policy</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Action Selection cost</td>
<td>(A^N)</td>
<td>(A)</td>
<td>(A)</td>
</tr>
</tbody>
</table>

Table 5.4: Comparison between different multi-robot control strategies.

5.8 Behavioral Difference

Diversity is a concept that has to do with differences, and with sub-goals. Assuming that the two robots have received the same information in the same kind of situations, they can still react differently. Sometimes, differences are due to different situation assessments, different utility and probability reasoning, and the impact of previous experiences. Balch [1998] has suggested a measurement for behavioral difference, then cluster the robots based on their behavioral differences in order to calculate hierarchical social entropy – this being the behavioral diversity measurement. Formally, behavior difference between two robots \(r_a\) and \(r_b\) is defined as:

\[
D(r_a, r_b) = \sum_i \left( \frac{p_a^i + p_b^i}{2} \right) |\pi_a(i) - \pi_b(i)|
\]  

(5.2)

Where \(\pi_j(i)\) is the \(r_j\)'s policy, \(p_j^i\) is the number of times \(r_j\) has encountered perceptual state \(i\) divided by the total number of times all states have been encountered and the hierarchical social entropy defined as:

\[
S(R) = \int_0^\infty H(R,h) \, dh
\]  

(5.3)

\(^{11}\) Due to the complexity of centralized control is exponentially with the number of robot. It is difficult to implement and could be used for systems consisting of many robots.
Where $R$ is the robots society under evaluation, $h$ is a parameter of the clustering algorithm indicating the maximum difference between any two robots in the same group and $H(R, h)$ is the simple entropy\footnote{The social entropy is calculated based on Shannon’s information entropy as following: $H(R) = -\sum_{i=1}^{N} p_i \log_2(p_i)$. Where, $R$ is a society of $N$ robots and $p_i$ is the proportion of robots in the $i$-th subset $\sum_{i=1}^{N} p_i = 1$.} of the society for the clustering at level $h$. Computing the behavioral diversity for the policies of the two foraging robots listed in Table 5.5, which are learned using the policy sharing, dynamic reward distribution and experience distribution, is 0.0, 0.29 and 0.22.

<table>
<thead>
<tr>
<th>Close to home</th>
<th>Target in gripper</th>
<th>Target visible</th>
<th>Two robots action</th>
<th>Response difference</th>
<th>Dynamic reward distribution</th>
<th>Response difference</th>
<th>Experiences distribution</th>
<th>Response difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 wander</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 acquire</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td>0.0 deliver deliver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 acquire</td>
<td>0.0 acquire deliver</td>
<td>1.0 wander deliver</td>
<td>1.0 wander deliver</td>
<td>1.0 wander deliver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 wander</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td>0.0 acquire acquire</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 acquire</td>
<td>0.0 acquire wander</td>
<td>1.0 wander wander</td>
<td>1.0 wander wander</td>
<td>1.0 wander wander</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 acquire</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td>0.0 wander wander</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: The learned policies of two foraging robots.

Unfortunately, the hierarchical social entropy of grouping robots may be impractical because it operates on a very low level (perceptual states), thereby making it a difficult scientific problem to sample and calculate. Moreover, it is not realistic to simplify a complex set of actions into a binary policy value; by doing this abstraction, almost all individual differences are filtered away, and the system might seem more homogenous than it actually is. Therefore, it is realized that a diversity index suggested by Balch [1998; 2000] is bounded to the time where the snapshot of the team was taken. The need for the a concept that captures the variations in the diversity over time, the flexibility of for increased or decreased diversity that is inherent in the system, and how this flexibility may be utilized in a rational way to increase robots’ team efficiency according to situations encountered. This concept is clear in the
shared policy learning strategy, where the diversity index equals zero, but I think it is not and the analysis of the state distribution for a number of robots using the same policy over some sufficiently long period of time may indicate behavioral differences between the robots, arising from variation of roles and different accumulation of experiences, this because the diversity of the system most often changes during execution.

5.9 Discussion and Conclusions

Reinforcement learning algorithms is a search process, in which the robot searches the environment for states that maximize the long-run reward and thus minimize the punishment. The time this search takes depends strongly upon the size and structure of state space and upon the a priori knowledge encoded in the learning robot’s initial parameters. When a priori knowledge is not available, the search through the state space is unbiased and can be excessively long. In natural, cooperative mechanism help to reduce the size of the search area, and hence the search time, by providing the learner(s) with auxiliary sources of bias (i.e., rewards and experiences). The ability of each robot to share its rewards and successful experiences offers a unique advantage – without explicit coordination of effort – the collective behavior can improve the individual’s performance [El-Telbany et al. 2002c]. Moreover, overcoming the problem faced in distributed reinforcement learning schemes of ignoring in the update rule what state or action they have been or taken. These ideas new algorithms their aims are to (1) speeding up learning and (2) enhancing the throughput performance. These aims based on:

- The training experiences are a form of planning gives an indication about global performance most beneficial direction by narrowing the search space.
- The cooperation rewards include information about other robot help to reduce the non-stationary in learning policies and the level of interference among them.

A fair comparison between multiple learning strategies using interference\(^{13}\) and speedup\(^{14}\) among robots as directly measurable property of multi-robot systems is presented. The interference among the robot teams is calculated by the difference among the actual team performance and their theoretical performance baseline. The results show that, the accumulated interference of the distributed experiences strategy drops off to half of the locally rewards

---

\(^{13}\) The interference and cooperation are two faces for one coin. As the interference increases the cooperation decreases and vice versa.

\(^{14}\) The speedup is considered the cooperation factor.
learning strategy. The flexibility of learning strategies is studied which refers to the ability of the robot team member to modify its behaviors as the environment or robot team changes. This includes the followings:

- The effect of adding non-learnable robots to the learned team on performance.
- The effect of adding non-learnable robots to the non-learned team on performance.

Finally, a comparison between the centralized and distributed learning strategies is presented to show their pros and cons. In addition, the learned policies’ diversity is measured to study the effect of learning on the policies knowledge. Unfortunately, the hierarchical social entropy of grouping robots may be impractical because:

- It operates on a very low level (perceptual states), thereby making it a difficult scientific problem to sample and calculate.
- It is not realistic to simplify a complex set of actions into a binary policy value; by doing this abstraction, almost all individual differences are filtered away, and the system might seem more homogenous than it actually is.

Therefore, it is realized that a diversity index suggested by Balch [1998; 2000] is bounded to the time where the snapshot of the team was taken. This leads to the need for another diversity index that captures the variations in the diversity over time and overcome the drawbacks listed above.
CHAPTER 6

MARKET-BASED REINFORCEMENT LEARNING FOR MULTI-ROBOT TASK-ALLOCATION

The problem that we are dealing with is how to make all robots learn to organize themselves by allocating their tasks dynamically. The dynamic task-allocation can be viewed as an action selection where a robot selecting a task to take so as to maximize its expected utility.

In this chapter, a novel approach for learning dynamic task allocation using a market-based reinforcement-learning algorithm is introduced. The learning algorithm based on economy model where robots have finite spending power and must choose what to spend it on, capable of learning this utility function for guiding the distributed planning process for task allocation process among multiple robots in multi-robot domains. Initial simulation results indicate the approach is successful at producing dynamic and effective task allocation for a team of several robots performing foraging and box pushing tasks.

6.1 Introduction

Multi-robot cooperation is an instance of dynamic task allocation or role allocation in distributed manner, which enables the system robust to failure and assists in the achievement of group(s) goal rapidly and optimizes tasks allocation/scheduling [Ostergaard et al., 2002]. In intentionally cooperative robotics; the dynamic task allocation can be viewed as an action selection problem [Maes, 1994; Humphrys, 1997; Ostergaard et al., 2002], where a robot is selecting a task to take so as to maximize its expected utility. The term utility can then be defined to any function reflecting the desires and priorities of the robot as shown in Figure 6.1.

For example:

- Parker [1994; 1998a] used the idea of dynamic task allocation in a decentralized multi-robot architecture by implemented an activation-threshold approach by mapped the robot’s ability to perform a task to a scalar quantity, which is used to assign tasks to robots. Using this method the team of robots has to be heterogeneous and each robot has to be characterized by a different threshold in order to regulate the activity of the team. She described multi-robot experiments with a priori hard-wired

---

15 The word task and role may be used one for the other
heterogeneous capabilities using ALLIANCE architecture, which has a mechanism of impatience and acquiescence, promoting robustness and fault tolerance. She used motivational behaviors to store information about other individual robots. Parker [1994, 1995b] extended her architecture by L-ALLIANCE implemented a style of parameters learning for adjusting activation thresholds used to perform dynamic task-allocation and task-ordering in her multi-robot system (cf. §4.2).

• Werger and Matarić [2000a; 2000b] provide broadcast of local eligibility (BLE) architecture, a distributive derivatives of Subsumption Architecture [Brook, 1986], to coordination among multiple robots by having the execution of the behavior on one robot directly inhabit the execution of the same behavior on another. The cross-inhabitation of behaviors is an opportunistic approach for distributing the tasks. In the BLE framework, robots have no commitment to their tasks; the relevant behaviors continuously communicate to decide who is best fit for each task, so that the tasks must be mutually exclusive.

• Gerkey and Matarić [2000; 2001a; 2001b; 2002] presents “Murdoch”; a completely distributed, resource-centric, publish/subscribe communication model. The approach is based on commitment, where a task allocation strategy using a market-based auction system commits the robots their tasks until success or failure. Firstly, each robot bids on a task based on its perceived fitness to perform the task. Secondly, a single round auction decides which robot gets the task. Finally, the winning robot’s controller performs a sequence of actions to execute the task. A key feature of this approach is that all communication is resource-centric and never name-based. On the side, the instantaneous greedy task scheduling in Murdoch does not allow for opportunistic optimization.

• Ostergaard et al., [2001, 2002], based on bidding mechanism, studied four different task allocation strategies, which derived from the combination of two variables: the amount of commitment to a given task engagement, and the amount of coordination among the robots.

Actually in task allocation, the robot’s ability to perform a task is mapped to a scalar quantity, where motivation and local eligibility are used to assign tasks to robots. Or by using an auction mechanism to decide which robot gets the task based on its bids values, which reflect their perceived strength to perform the task. The auction mechanism use the economy model, where robots have finite spending power and must choose what to spend it on or try to maximize
its utilities based on revenue and cost function across possible plans for executing a specific task [Ostergaard et al., 2002; Dias and Stentz; 2000; Zlot et al., 2002; Gerkey and Matarić, 2002].

![Diagram of a robot with different parts highlighted and connected by arrows.](image)

**Figure 6.1:** Learning dynamic task allocation based on can be viewed as an action selection problem where a robot is selecting a task to take so as to maximize its expected utility (Adopted from Gerkey and Matarić, 2002).

However, the learning to dynamically allocate the tasks among a group of multiple robots is as yet unsolved. Learning to allocate tasks is depended on the robots’ ability to handle the task, their experience in the same task, progress in task execution, their metric distance to the task, their expected profit if they doing this task, and so on. This information can be learned by some sort of reinforcement learning algorithms and guided the distributed planning process for dynamic task allocation process. The traditional temporal reinforcement learning [Sutton and Barto; 1998] formulizes the problem of learning from interaction with an environment. The two-standard approaches to reinforcement learning are “value iteration” and “policy iteration”. These standard approaches depend on enumerating state space, and are problematic when the state space is huge. This is known as the curse of dimensionality problem. The convergence theorem for them required infinite sapling size as will as strong (often Markovian) assumption about the environment. Moreover, they depend on finding an evaluation function shown progress after a single action, and such a function may be extremely hard to learn, or even to represent.

---

16 The curse of dimensionality leads to the problem of deciding how the different aspects of an input affect the value of the output. This is called also the “structure credit-assignment” problem. The structure credit-assignment is solved by generalization the state space by approximating the value function or Q-function.
In this chapter, an approach to learning multi-robot task-allocation is presented, in simulation. The proposed approach focuses on the use of market-based reinforcement learning which based on the economical model, where robots team try to allocate the tasks that maximize its utilities based on its previous experience and cost function, which in principle does not suffer from the limitations of traditional reinforcement learning algorithms. In addition, its highly robust to changes in the environment including malfunctioning robots, successful robots are able to accumulate wealth (i.e., experiences) and perpetuate their winning strategies [Dias and Stentz, 2000]. The chapter will address the following problem: how can dynamically allocate tasks among multiple robots using learning? a market-based reinforcement learning in which the multi-robot teams dynamically learn task allocation based on their experiences and perceived information is proposed. Demonstrating an effective learning algorithm on a group of robots learning on forage and box pushing to validate these ideas. Table 6.1 summarizes the problem and the approach.

### 6. Market-Based Reinforcement Learning for Task-Allocation

The previous surveyed examples there is no discussion about on-line adaptation for task allocation is presented. In this section, a proposed approach focuses on the use of market-based reinforcement learning is designed to meet this criterion for task-allocation among robot teams. The robots interact with their environment and communicate with each other to allocate tasks that maximize its utilities based on its previous experience and cost function, which in principle does not suffer from the limitations of traditional reinforcement learning algorithms. Each robot bids on an announced task based on its experience and perceived

<table>
<thead>
<tr>
<th>Problem</th>
<th>Learning to allocate tasks among multiple robots.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion</td>
<td>Tasks must allocate dynamically based on learning experiences and perceived information.</td>
</tr>
<tr>
<td>Approach</td>
<td>Formulate a centralized and distributed bidding algorithm as market-based reinforcement learning.</td>
</tr>
<tr>
<td>Validation</td>
<td>Implementing learning on a group of mobile robots learning to forage as homogenous task and to push-boxes with forage as heterogenous task to explore the centralized and distributed allocation strategies.</td>
</tr>
</tbody>
</table>

Table 6.1: A summary of the situated learning problem addressed here, and the structure of the proposed solution.
capability to doing the task. The auction robot selects its best candidate based on its in submitted bids value, and send a request for doing a task for the winner, which win the competition for the right to carry out this task, that has the maximum value function and perceived capability to doing the task $T$ as follows:

$$\max_i \left( v_i(T) + c_i(T) \right)$$  \hspace{1cm} (6.1)

Where $v_i(T)$ is a task motivation value function and $c_i(T)$ is a perceived capability for robot’ task $T$. For example, the distance cost or energy cost can be used as a perceived capability for reaching the task. The bidding gain when the robot will carry out the task request $b_i(T)$ is defined by:

$$b_i(T) = k \cdot \left( v_i(T) + c_i(T) \right)$$  \hspace{1cm} (6.2)

Where $k$ is bidding-rate, a real number ($0 < k < 1$), and represent the rate that get the deterministic part of bid that will possibly lost. These bidding values are collected by communication with other robots. Then selecting the highest bidding value from winning robot from this competition such that:

$$b_i(T) = \max_k \{b_k(T)\}$$  \hspace{1cm} (6.3)

By making this robot be active and take the right to carry out assigned task. Credit assignment between robots by adjustment of the collected experience in the value function of task-achieving behavior $v_i(T)$ of the winner by reducing it with the bidding value as follow:

$$v_i(T) = v_i(T) - b_i(T),$$  \hspace{1cm} (6.4)

and hands this amount back to the losers, which in turn adds the received amount to its value function as follows:

$$v_j(T) = v_j(T) + \frac{b_i(T)}{|n|}$$  \hspace{1cm} (6.5)
Where \( |n| \) is the number of loser' robots. After the winner carry out the task, receive an external reward \( r_i \) and modify its motivation value function according to:

\[
v_i(T) = v_i(T) + k_o \cdot r_i(T)
\] (6.6)

The joint effects of all of these updates are described by the following rule:

\[
v_i(T) = (1 - k_0) \cdot v_i(T) + k_0 \cdot (r_i(T) - c_i(T))
\] (6.7)

This motivation value function \( v_i(T) \) is bounded by maximum and minimum value of the summation \( r_i(T) - c_i(T) \) (The proof of bounded is shown in Appendix C).

The proposed algorithm has the following advantages:

- Presents an on-line adaptation methodology for task allocation.
- Learning process tries to maximize the utility values using the previous experience and cost function.
- Learning algorithm based on the economical model which uses an auction mechanism to decide which robot gets the task.
- Overcomes the standard reinforcement learning approach, which depends on enumerating state space, Markovian assumption.

### 6.3 Experiments Validation

In order to validate the proposed approach, and demonstrate how the market-based reinforcement learning and communication dynamically allocate tasks, a variant of emergency handling as our problem task domain for evaluation is used [Ostergaard et al., 2001; 2002]. Instead of alarms occur at unpredictable times in an office environment and a group of robots teams tries to detect and fix problems indicated by those alarms, in our version the group of robots, where they are doing a foraging task, a failures for some robots occurs at unpredictable times that broadcast a call for help to non-failure robots. The task of the robot team is to fix the failure robots by respond to the call for help, and by restrict us to the case where any robot can fix any robot’s failure. Deciding which robot should go and respond to the call for help, is a key problem. The solution is depends the robots’ ability to handle the
failure, their metric distance to the failure robot, the previous level of experience on fixing this failure, the expected reward from the failure robot, and so on. A two tasks are explored which are:

1. Allocation strategies in the context of market-based reinforcement learning. This depend on the amount of coordination among the robots, highly-coordinated or centralized strategy where no two robots were allowed to engaged in the same task at the same time and the blackboard evaluated by a centralized controller, and uncoordinated or distributed strategy where the robots were allowed to engaged in the same task at the same and each robot evaluates the blackboard.

2. Heterogeneity effect on task-allocation is explored by using different task domain. In homogenous task the robots have the same behavior sets and heterogeneous task the robots have different behaviors sets.

6.4 Experiments Design

To explore the effect of the amount of coordination and the heterogeneity on learning dynamic task allocation as discussed in previous section, we designed experiments where each robot learns to fix the failures emergent by call for help messages and dynamically decide to proceed or response to the call based on centralized or distributed strategy. There are two task domains are used foraging task and mixing box-pushing with foraging task as shown in Figure 6.2. In the foraging task the robots are homogenous but in the box-pushing with foraging task the robot are heterogenous, such that they doing different tasks. All the simulated experiments consist of the $10m \times 10m = (100m^2)$ world. The world of foraging task contains 20-attractors, 12 of them are stationary and the others are movable and for box-pushing and foraging task an additional 4 boxes are added. In additional the field includes nine obstacles varying from about $0.5m^2$ to $1m^2$ coverage approximately 5% of the field. Additionally, there are six robots was used in each experiment. In foraging task, the robots have the same behavior sets but in box-pushing and foraging task the robots have different behavior sets. In each simulation run, robots are allowed to learn dynamic-task allocation until an episode of 50 cycles of 10 second is elapsed. The bidding rate parameter for the experiments were set as $k_0 = 0.05$ and the reward of doing the task $r = +5$, which is taken when failure is fixed.
6.4.1 Case I: Centralized Task-Allocation Among Homogenous Robots (CHM)

In our first experiment implementation, the foraging task is extended by with designed six abstract movement behaviors: *wander, deliver, acquire, stay, repair*, and *go-to-failure* (cf. Appendix B). These six behaviors made use of lower level capabilities, such as obstacle avoidance. On the same level of the control behaviors, we implement six communication behaviors: *auction, evaluate, submit, announce*, and *credit*. The communication behaviors learn the dynamic task-allocation process, through *centralized* market-based reinforcement-learning algorithm, where the communication processes are done thought blackboard and there is a centralized robot allocate the tasks. Using binary perceptual features that represent the appropriate robot state space that are used to sequence the robot through steps in achieving the task. In the task each robot has the perceptual features listed in Table 6.2. The robot must decide on the basis of these environmental cues which behavior to activate at each point in time. The appropriate motor-schema corresponding to actions in achieving this task described and listed in Table 6.3.
Table 6.2: Description of the perceptual schemas designed and used in centralized task-allocation in foraging task.

| **at_home?** | Check the state of the robot if it is at home or not. |
| **target_visable?** | Check the state of the robot if it encounters an attractor or not. |
| **have_target?** | Check the state of the robot if it grasps the attractor or not. |
| **failure_occur?** | Check the state of the robot if it in failure or not. |
| **oculation_open?** | Check the state if an auction is opened by failure robot or not. |
| **bid_evaluated?** | Check the state if submitted bids are evaluated or not by failure robot. |
| **bid_submitted?** | Check the state if a bid is submitted by repairing robot or not. |
| **failure_repaired?** | Check the state of the robot if its failure is repaired or not. |
| **at_failure?** | Check the state of the repairing robot if it is at failure or not. |
| **is_winner?** | Check the state of the repairing robots if selected as winner or not. |

Table 6.3: Description of the motor-schemas designed and used in centralized task-allocation in foraging task.

| **wander** | Move randomly about the environment in search for attractors. |
| **acquire** | Move towards the closest visible attractor. |
| **deliver** | Move towards the delivery area. |
| **stay** | Stay at the current position. |
| **repair** | Repair the failure robot. |
| **go_to_failure** | Go to the failure robot. |
| **auction** | Ask for help and open an auction on the blackboard. |
| **submit** | Submit your bid based-on your experience and capability to blackboard. |
| **evaluate** | Evaluate submitted bids to select the winner by centralized controller. |
| **announce** | Notify the winner for doing the repair by centralized controller. |
| **credit** | Credit adjust based on you are winner or loser by centralized controller. |

Where, the control activation uses the finite state automaton (FSA) descriptions for sequenced behaviors as shown in Figure 6.3, which contains the main parts of our centralized control architecture.
6.4.2 Case II: Distributed Task Allocation Among Homogenous Robots (DHM)

In our second experiment implementation, the foraging task is extended with designed six abstract movement behaviors used in the previous section (cf. §6.4.1), and replacing the communication behaviors by two new behaviors: auction, and evaluate. The communication behaviors learn the dynamic task-allocation process, through distributed market-based reinforcement-learning algorithm. In this strategy each robot decide by himself thought evaluating the blackboard to allocate the tasks, and permits the robots to engage in the same task at the same time as shown in Figure 6.4. In the task each robot has the perceptual features listed in Table 6.4. The appropriate motor-schema corresponding to actions in achieving this task described and listed in Table 6.5.

<table>
<thead>
<tr>
<th>at_home?</th>
<th>Check the state of the robot if it is at home or not.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_visable?</td>
<td>Check the state of the robot if it encounters an attractor or not.</td>
</tr>
<tr>
<td>have_target?</td>
<td>Check the state of the robot if it grasps the attractor or not.</td>
</tr>
<tr>
<td>failure_occur?</td>
<td>Check the state of the robot if it in failure or not.</td>
</tr>
<tr>
<td>announce_for_tasks?</td>
<td>Check the state if a new task is announced by failure robot or not.</td>
</tr>
<tr>
<td>failur_repaired?</td>
<td>Check the state of the robot if its failure is repaired or not.</td>
</tr>
<tr>
<td>at_failure?</td>
<td>Check the state of the repairing robot if it is at failure robot or not.</td>
</tr>
<tr>
<td>is_winner?</td>
<td>Check the state of the robot if winning the competition or not.</td>
</tr>
</tbody>
</table>

Table 6.4: Description of the perceptual schemas designed and used in distributed task-allocation in foraging task.
Figure 6.4: The distributed control architecture permits the robots to engage in the same task at the same time. In the upper-left corner two robots respond to call-for-help and went to the failure robot simultaneously.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wander</td>
<td>Move randomly about the environment in search for attractors.</td>
</tr>
<tr>
<td>acquire</td>
<td>Move towards the closest visible attractor.</td>
</tr>
<tr>
<td>deliver</td>
<td>Move towards the delivery area.</td>
</tr>
<tr>
<td>stay</td>
<td>Stay at the current position.</td>
</tr>
<tr>
<td>repair</td>
<td>Repair the failure robot.</td>
</tr>
<tr>
<td>go_to_failure</td>
<td>Go to the failure robot.</td>
</tr>
<tr>
<td>auction</td>
<td>Ask for help on the blackboard.</td>
</tr>
<tr>
<td>evaluate</td>
<td>Submit your bid to blackboard and evaluate the submitted bids to decide if you are winning the competition or not.</td>
</tr>
</tbody>
</table>

Table 6.5: Description of the motor-schemas designed and used in distributed task-allocation in foraging task.
6.4.3 Case III: Centralized Task-Allocation Among Heterogeneous Robots (CHT)

In our third experiment, a box-pushing task is implemented by designing abstract movement behaviors: *wander*, *push-box* and *adjust*. In addition, the same communication behaviors designed in centralized strategy of foraging task is used (cf. §6.4.1). For the box-pushing task each robot has the different perceptual features listed in Table 6.6, and motor-schema corresponding to actions in achieving this task described and listed in Table 6.7. In the experiment, heterogeneous robots are simulated, to see the effect of behaviors heterogeneity on task-allocation process performance.

<table>
<thead>
<tr>
<th>at_home?</th>
<th>Check the state of the robot if it is at home or not.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_visable?</td>
<td>Check the state of the robot if it encounters a box or not.</td>
</tr>
</tbody>
</table>

**Table 6.6: Description of the perceptual schemas used in centralized task-allocation in box-pushing task.**

<table>
<thead>
<tr>
<th>wander</th>
<th>Move randomly about the environment in search for boxes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>push-box</td>
<td>Push a visible box to the home.</td>
</tr>
<tr>
<td>adjust</td>
<td>Adjust your position with respect to the box and home positions.</td>
</tr>
</tbody>
</table>

**Table 6.7: Description of the motor-schemas used in centralized task-allocation in box-pushing task.**

6.4.4 Case IV: Distributed Task-Allocation Among Heterogeneous Robots (DHT)

In our fourth experiential implementation, the behaviors described in distributed strategy are used (c.f. 6.4.2). The experiment simulates heterogeneous robots and doing foraging and box-pushing tasks.

6.5 Experiments Results

Evaluating our market-based reinforcement-learning algorithm for distributed and centralized dynamic task-allocation, a data from twenty trials for six robots was collected from the four cases explained in previous section (cf. §6.4) and averaged. The comparison among these four cases is based on the following performance metrics:
1. **Average response time**, which is the average response time needed by the team to converge to stable policy over time. The response time is measured by calculating the time between the task-allocation and task-repair.

2. **Success rate**, which was measured the number of repaired failures from allocated task. This is measured by the percentage allocated tasks to announced tasks. In other word, the percentage of responding to the call-for-help messages.

The rate at which the robots converge to stable policy is evaluating by tracking the average amount of time needed by the team to converge to stable response time during each trial. Plotting the average response time to measure the learning curves of the four task allocation cases is shown in Figure 6.5. In the four cases the average time of between the task-allocation and task-repair is reached a stable level due to learning process of failure task-allocation. So, all the four cases show good convergence properties. The robots’ team that used the distributed allocation strategy converges faster than centralized allocation strategy. This may be explained due to the centralized allocation strategy takes more times in processing the tasks’ announcement and resolving the conflicts among eligible robots based on their identification number (ID), where the less the ID number the chance to be the winner.

![Figure 6.5: The average response time needed by the team to converge to stable policy over time of robots' team in the four cases.](image)

The distributed allocation strategy has minimum average response time this can be explained due to in many instances of time many robots respond to call-for-help and went to the failure
robot simultaneously and the nearest robots in these situations done the repairing process and compensate the experience. The distributed allocation strategy performs better than centralized strategy. From Figure 6.5, the team’s average response time for different system is shown. The distributed strategies have the minimum average team’ response time, however, it can be seen that there is no statistically significant difference between homogenous team and heterogeneous team for different coordination strategy. The difference in average time of response is small but can be explained due to the difference of execution time of heterogonous behaviors. So, heterogeneity in behaviors adds extra overhead on the average response time. Counting the number of success of repaired failures done by each robot for different case is plotted in Figure 6.6. It can be seen that the heterogeneous teams have higher success rate than homogenous teams. Moreover, there is no statistically significant difference between homogenous team and heterogeneous team for different coordination strategy. Theses finding nearly corresponding with the average response time data.

![Graph](image)

**Figure 6.6:** The success rate of robots team in the four cases; this is measured by the percentage allocated tasks to announced tasks

The execution sequences depicted in Figure 6.7 and Figure 6.8 shows task-allocation process using market-based reinforcement learning in a homogenous and heterogeneous robot teams using distributed coordinating strategy, in the early stages the task allocation process is random until the robots gains the sufficient experiences to organize themselves and takes the responsibilities of allocation of submitted tasks.
Figure 6.7: This sequence of snapshots shows the progress of learning process of task-allocation in homogenous robot’s team.
Figure 6.8: This sequence of snapshots shows the progress of learning process of task-allocation in homogenous robot's team.
6.6 Discussion and Conclusions

In this chapter, a novel approach for learning task-allocation among multiple robots is proposed and demonstrated. The reinforcement learning approach is based on market model, which named as market-based reinforcement-learning algorithm. The algorithm uses a communication mechanism among the robots for aiding various aspects of learning in distributed multi-robot system. The proposed algorithm has the following advantages:

- Presents an *on-line adaptation* methodology for task allocation.
- Learning process tries to maximize the utility values using the *previous experience* and *cost function*.
- Learning algorithm based on the economical model which uses an *auction mechanism* to decide which robot gets the task.
- Overcomes the standard reinforcement learning approach, which depends on enumerating state space, Markovian assumption.

The effect of the market-based reinforcement learning has explored on the performance of coordination and heterogeneity of robots’ team in the context of four different cases:

1. Centralized Task-Allocation among Homogenous Robots (CHM)
2. Distributed Task-Allocation among Homogenous Robots (DHM)
3. Centralized Task-Allocation among Heterogeneous Robots (CHT)
4. Distributed Task-Allocation among Heterogeneous Robots (DHT)

The results show that:

- All the four cases show good convergence prosperities.
- The distributed allocation strategy performs better than centralized strategy due to in many instances of time many robots respond to call-for-help and went to the failure simultaneously.
- Heterogeneity in behaviors adds extra overhead on the average response time.

The future work is to replace the FSA control with a *Q-learning* adaptive control. These two layers form a coordination and organization capabilities for robot control.
CHAPTER 7

CONCLUSIONS

In this dissertation, a new focus for research in behavior-based multi-robot reinforcement learning is provided. Our focus on the problems of coordination and organization in cooperative robots are from two perspectives.

- First, focuses on behavior-based robots control, in which the robots behaviors can be programmed, are presented.
- Second, novel algorithms involved coordinating and dynamic task-allocation among homogenous multi-robots systems are presented.

In this chapter, the dissertation’s results are summarized and the future works are presented.

7.1 Summarizing the Results

Previous research in multi-robot reinforcement learning has identified as promising for building self-organizing that can capable to coordinate themselves in real-time tasks. This dissertation furthers the understanding of multi-robots systems by examining particular aspects not addressed by previous work. This work can be summarized as follows:

1. An introduction to the field of behavior-based control includes the motivation for using behavior-based control and comparison between the most used behaviors control methodologies is presented identifying their strengths as well as weaknesses (cf. chapter 2).
2. Next a reinforcement learning algorithms are introduced and the other reinforcement learning implemented in the behavior-based robotic control are discussed by highlighting the pros and cons of these methods as a motivation for using temporal-difference reinforcement learning. After introducing the field of behavior-based control and reinforcement learning, a review of cooperative robotics is presented to identifying the methods that incitement the multiple robots to cooperate. In addition, the recent research for integrating reinforcement learning and cooperative multi-robot systems is presented (cf. chapter 3).
3. Following these reviews, a focus on the experimental work of this dissertation is done. The experiments focus on learning coordination and dynamic task allocation
among cooperative multi-robots. A framework for distributed reinforcement learning for learning from distributed rewards is proposed where the robots cooperate by sharing their rewards and introducing new algorithms for estimating the distributing parameters which determine the percentage of sharing to the overall reward (cf. chapter 4). The proposed approach have the following advantages:

- Provides auxiliary source of bias.
- Learning from stable reward function.
- Adaptive estimation of weights function.

Moreover, a comparison between centralized and distributed control in multi-robot system is presented to show their pros and cons to contrast the proposed approach.

4. Extends the previous works by distributing both of successful expertise and rewards into the framework of *apprentice learning* are presented where the robots learned from others. The new learning algorithms aims to (1) *speeding up learning* and (2) *enhancing the throughput performance* based on:

- The training experiences are a form of planning gives an indication about global performance most beneficial direction by *narrowing the search space*.
- The cooperation rewards include information about other robot help to reduce the *non-stationary* in learning policies and the level of *interference* among them.

The results of the experiments demonstrate superior performance for distributed reward and experiences, strong performance for distributed experiences and good performance for distributed rewards. A comparison between multiple learning strategies using *interference* and *speedup* among robots as directly measurable property of multi-robot systems is presented. The interference among the robot teams is calculated by the difference among the actual team performance and their theoretical performance baseline. The results show that, the accumulated interference of the distributed successful experiences strategy drops off to half of the locally rewards learning strategy. The *flexibility* of learning strategies is studied which refers to the ability of the robot team member to modify its behaviors as the environment or robot team changes. This includes the followings:

- The effect of adding *non-learnable* robots to the learned team on performance.
• The effect of adding non-learnable robots to the non-learned team on performance.

Finally, a comparison between the centralized and distributed learning strategies is presented to show their pros and cons. In addition, the learned policies’ diversity is measured to study the effect of learning on the policies knowledge. Unfortunately, the hierarchical social entropy of grouping robots may be impractical because:

• It operates on a very low level (perceptual states), thereby making it a difficult scientific problem to sample and calculate.

• It is not realistic to simplify a complex set of actions into a binary policy value; by doing this abstraction, almost all individual differences are filtered away, and the system might seem more homogenous than it actually is.

Therefore, it is realized that a diversity index suggested by Balch [1998; 2000] is bounded to the time where the snapshot of the team was taken. This leads to the need for another diversity index that captures the variations in the diversity over time and overcome the drawbacks listed above (cf. chapter 5).

5. Next, a novel approach for learning task-allocation among multiple robots is proposed and demonstrated by designing the motor and perceptual schemas. The reinforcement learning approach is based on market model, which named as market-based reinforcement-learning algorithm. The algorithm uses a communication mechanism among the robots for aiding various aspects of learning in distributed multi-robot system. The proposed algorithm has the following advantages:

• Presents an on-line adaptation methodology for task allocation.

• Learning process tries to maximize the utility values using the previous experience and cost function.

• Learning algorithm based on the economical model which uses an auction mechanism to decide which robot gets the task.

• Overcomes the standard reinforcement learning approach, which depends on enumerating state space, Markovian assumption.
The effect of the market-based reinforcement learning has explored on the performance of coordination and heterogeneity of robots’ team in the context of four different cases:

- Centralized Task-Allocation among Homogenous Robots (CHM)
- Distributed Task-Allocation among Homogenous Robots (DHM)
- Centralized Task-Allocation among Heterogeneous Robots (CHT)
- Distributed Task-Allocation among Heterogeneous Robots (DHT)

The results show that:

- All the four cases show good convergence prosperities.
- The distributed allocation strategy performs better than centralized strategy due to in many instances of time many robots respond to call-for-help and went to the failure simultaneously.
- Heterogeneity in behaviors adds extra overhead on the average response time.

The future work is to replace the FSA control with a Q-learning adaptive control. These two layers form a coordination and organization capabilities for robot control (cf. chapter 6).

### 7.2 Issues for Future Research

This work is intended as a foundation in a continuing effort toward studying increasingly complex social robots capable of more complex learning, and through it, more complex intelligence. Important future work includes:

- Using different cooperative distributing functions, in order to explore their effect on the performance.
- Implementing the new proposed algorithms for other homogenous and heterogeneous multi-robot tasks.
- Extension of distributed reinforcement framework by theoretical analysis, in order to proof the effectiveness mathematically in addition to our experiment results.
• Extended the market-based reinforcement learning task-allocation model by replacing the FSA control with a \textit{Q-learning} adaptive control. These two layers of learning form a coordination and organization capabilities for robot control.
A P P E N D I X  A


A.1 Behavior Encoding

A behavior is a mapping of sensory inputs to a pattern of motor actions, which then are used to achieve a task as shown in Figure A.1.

![Figure A.1: Simple Behavior Diagram.](image)

To encode behavior response that the stimuli evoke, a functional mapping is created from the stimuli plane to the motor plane. By factoring the robot’s motor response into orthogonal components: strength and orientation. Strength denotes the magnitude of the sensory input and orientation denotes the direction of action. The behavior can express as triple \((S, R, \beta)\) where \(S\) denotes the domain of all interpretable stimuli, \(R\) denotes the range of possible response, and \(\beta\) denotes the mapping \(\beta : S \rightarrow R\).

\(S\) consists of all the perceivable stimuli. Each individual stimulus or percept \(s \in S\) is represented as a binary tuple \((p, \lambda)\) having a particular type or perceptual class \(p\) and a property of strength \(\lambda\). The stimulus strength \(\lambda\) can be defined in a variety of ways: discrete (e.g., binary: absent or present; categorical: absent, weak, medium, strong) or real valued or continuous. The instantaneous response \(r \in R\) of the mobile robot that moves on flat ground and can rotate only about its central axis has three degree of freedom expressed as \(r=[x, y, \theta]\). For each active behavior we can formally establish a mapping between stimulus and response using the behavior function \(\beta(s)=r\). Associated with a particular behavior, \(\beta\), may be a scalar gain value \(g\) modifies the magnitude of the overall response \(r\) for a given \(s\) as: \(r' = gr\).
These gain values are used to compose behaviors by specify their strengths relative to one another. In the extreme case, \( g \) can be used to turn off a behavior by setting it to 0, thus reducing \( \beta \) to zero [Arkin, 1998].

\( \beta \) is defined arbitrary, but it must be defined over all relevant \( p \) in \( S \). The behavior mappings, \( \beta \), fall into three general categories:

- **Null**: The Stimulus produces no motor response.
- **Discrete**: The stimulus produces a response from an enumerative set of prescribed choices (e.g., turn-right, stop… etc.). For example, \( \beta \) consists of a finite set of (situation, response) pairs. Sensing provides the index for finding the appropriate situation. Another strategy involves the use of a rule-based system. Here \( \beta \) is represented as a collection of IF-THEN rules [Matarić, 1994], which can be extended using the fuzzy logic by synthesis a Fuzzy IF-THEN rules [Saffiotti, 1997; Hoffmann, 1998].
- **Continuous**: The stimulus produces a response that is continuous over \( \mathbb{R} \)’s range. One of the most common methods for implementing continuous response is based on the potential field technique (cf. §2.3).

### A.2 Concurrent Behaviors Combination

Having discussed method to describe individual behaviors, we now study methods for constructing systems consisting of multiple behaviors, where multiple behaviors may be concurrently active with the robot system. The behavioral coordination function, \( C \), is now defined such as \( \rho = C(G \ast B(S)) \) or \( \rho = C(G \ast R) \). \( C \) can be arbitrarily defined, but several strategies are commonly used to encode this function. They are split across two dimensions: competitive and cooperative. The simplest competitive method is pure arbitration, where only one behavior’s output (\( r \)) is selected from \( \mathbb{R} \) and assigned to \( \rho \), what is, arbitrarily choosing only one response from the many available. Several methods used to implement this particular technique, including behavioral prioritization (Subsumption). Cooperative methods, on the other hand, blend the outputs of multiple behaviors in some way consistent with the robot’s overall goal. The most common method of this type is vector addition or super-positioning.

Competitive and cooperative methods can be composed as well [see Arkin, 1998 for more details]. This is illustrated in Figure A.2.
A.3 Subsumption Architecture Behaviors’ Programming

Task-achieving behaviors in the subsumption architecture are represented as separate layers. Individual layers work on individual goals concurrently and asynchronously. Where each behavior is represented by augmented finite state machine (AFSM) model (cf. §2.2). This method is difficult in design and implementation. Brooks [1990c] recognize the problem and develop the Behavior Language, which provide a new abstraction independent of the AFSMs using a single rule set to encode each behavior. This high level language is then compiled to the intermediate AFSM representation, which can then be further compiled to run on arrange of target processors.

![Behaviors Coordination Techniques](image)

**Figure A.2: The classification of multiple behaviors coordination techniques (adopted from Pirjanian, 1999).**

To illustrate a programming example how the subsumption-based design is programmed. We design and implement a foraging example for a robot having *two sonar sensors* on the front of its body. Each behavior in the system is encoded as a set of rules (standard for the Behavior Language) [Arkin, 1998]. Before programming of the behavior begins the priorities of the behaviors have to be thought. It looks like a hard problem since the many layers of behaviors are not dividing work and occasionally pass results to each other. Usually the lowest level of behavior is programmed first and tested on the robot until it works. When the behavior is working it is frozen and programming of the behavior of the next level is initiated. The new behavior is tested together with the previous behavior and adjusted until every thing is works. This process is continued until the desired level of competence is achieved. The proposed architecture is shown in Figure A.3.
Coding theses behavior is straightforward for example, a wander behavior which making the robot to move in a random direction for some time has the following Java code segment.

```java
private int wander() {
    if (tt > 5) {
        tt--;
        move_random();
    }
    return 0; /* set priority = 0 which activate find_goal behavior */
}
```

For the pickup behavior, which turn towards the sensed attractor and go forward. If at the attractor, close gripper, has the following Java code segment.

```java
private int pickup() {
    int dif;
    if (obstacle) {
        return 1; /* set priority = 1 which activate avoid_obstacles behavior */
    }
    if (pellet) {
        pellet = false;
        return 0; /* set priority = 0 which activate find_goal behavior */
    }
    if (energy < energy_low) {
        return 3; /* set priority = 3 which activate homing behavior again */
    }
    dif = laser_sen[0] - laser_sen[1]; /* calculate the difference between right and left laser sensors */
    if (dif >= -40 && dif <= 40) { move_forward(); }
    if (dif < -40) { move_left(); }
    if (dif > 40) { move_right(); }
    return 2; /* set priority = 2 which activate pickup behavior again */
}
```
Coordination the multiple behaviors based on behavioral prioritization. Each behavior has a priority value and every behavior execute its action if its priority matches the current priority variable value, this can be coded as following:

```java
priority = 0;
public void run() {
    while(isAlive()) {
        switch(priority) {
            case 0: priority = find_goal(); break;
            case 1: priority = avoid_obstacles(); break;
            case 2: priority = pickup(); break;
            case 3: priority = homing(); break;
            case 4: priority = upload(); break;
            case 5: priority = recharge(); break;
            default: priority = wander(); break;
        }
    }
}
}
```

A.4 Behaviors and Schema Theory

Schemas were conceived of by psychologists as a way of expressing the basic unit of activity. A *schema* consists both of the knowledge of to act and/or perceive as well as the computational process by which it is used to accomplish the activity. The idea of schema maps very well onto a class in object-oriented programming. Extending the schema theory towards a computation theory of intelligence, the building block for robot intelligence, is equivalent to a schema and composed of a *motor schema* and a *perceptual schema*. The motor schema represents the template for the physical activity; the perceptual schema embodies the sensing as shown in Figure A.4.

![Figure A.4: Behavior decomposed into perceptual and motor schema (adopted from Murphy, 2000).](image-url)
The schema theory implemented the motor schema as a *vector field*, which direction and magnitude of the action. In the schema theory, the perceptual schema is permitted to path both the percept and a gain to the motor schema. The motor schema can use the gain to compute a magnitude on the output action. However, schema theory does not specify how the output from concurrent behaviors is combined. Where the output of concurrent behaviors in some circumstances is combined or summed, in others occur in a sequence, and sometimes would be the winner-talk-all effect.

### A.5 Motor-Schema Behavior Programming

The motor-schema behavior-based methodology based on the potential fields. The potential fields are actually easy to program, especially since the fields are *egocentric* to the robot. The robot computes the effect of the potential field, usually as a straight line, at every update, with no memory of where it was previously or where the robot has moved. There are five basic potential fields, or primitives, which can be combined to build more complex fields: *uniform, perpendicular, attractor, repulsive, and tangential* as shown in Figure A.5.

![Figure A.5: Five primitive potential fields (a) uniform, b) perpendicular, c) attractive, d) repulsive, and e) tangential (adopted from Murphy, 2000).](image)

Any primitive potential field is usually represented by a single function. The vector impacting the robot is computed each update. Consider a robot with a single range sensor facing forward. The designer has decided that the repulsive field with a liner drop off is appropriate. The formula is:
\begin{align*}
V_{direction} &= -180^\circ \\
V_{magnitude} &= \begin{cases} 
\frac{(D-d)}{D} & \text{for } d \leq D \\
0 & \text{for } d > D 
\end{cases} 
\end{align*} \tag{A.1}

Where \( D \) is the maximum range of the field’s effect, or the maximum distance at which the robot can detect the obstacles. \( D \) is always the range at which the robot should respond to stimulus. The roboticist set a \( D \) of 2 meters and the robot maximum velocity to 10 meters. This can be coded using Java as follow:

class vector {
    double  magnitude;
    double  direction;
    public vector() {
        magnitude = 0;
        direction = 0;
    }
}

class perceptual_schema {
    double readSonar ()    {    /* hardware dependent function */ }
}

class motor_schema {
    public static final double MAX_DIST = 2.0;
    public static final double MAX_VELOCITY = 10.0;
    vector repulsive (double d, double D) {
        vector output = new vector();
        if (d <= D) {
            output.direction = -180;
            output.magnitude = (D – d)/D;
        } else return (new vector()); /* stay where you are */
    }
}

To illustrate how the repulsive potential field can be used by a behavior, runaway for a robot with a single sensor as a motor-schema. The following code segment clarifies the idea.

class runaway extended Behavior {
    perceptual_schema  perceptual;
    motor_schema  motor;
    public runaway (perceptual_schema p, motor_schema  m) {
        perceptual = p; /* construct the runaway behavior */
        motor = m;
    }
}
vector execute_runaway() {
    /* every time step get perceptual value and calculate the potential field */
    double reading = perceptual.readSonar();
    vector output = motor.repulsive(reading, motor_schema.MAX_DIST);
    return (output);
}

The robot can execute the runaway behavior as soon is became on as follow:

vector output;
While (isAlive()) {
    output = execute_runaway();
    turn (output.direction);
    forward(output.magnitude * motor_schema.MAX_VOLICITY);
}

The implementation of runaway behavior for a robot with a single sensor is straightforward. But what if the robot has multiple range sensors? Multiple sensors will detect bigger obstacles at the same time. The common way is to have a runaway behavior for each sensor. This called multiple instantiation of the same behavior and using the vector summation prosperity of potential field to generate a single vector value.

A.6 Motor-Schema Concurrent Behaviors Combination

A robot will generally have forces acting on it from multiple behaviors. In motor-schema concurrent behaviors is combined and coordinated by vector summation. Each behavior’s relative strength or gain, \( g_i \), is used as a multiplier of the vectors before addition. Figure A.6 illustrate how this is accomplished.

![Figure A.6: Behavior Fusion via vector summation (adopted from Arkin, 1998).](image-url)
For example, adding another behavior, *move_to_goal* that is represented with an attractive potential field and uses the shaft encoders to sense if reach the goal position. The *move_to_goal* behavior exerts an attractive field over the entire space; wherever the robot is, it will feel a force from the goal. The runaway behavior exerts a repulsive field in a radius around the obstacle. Extending our example in the previous section with the two behaviors coded in java as follows:

```java
vector output, vec1, vec2;
While (isAlive())
{
    vec1 = execute_runaway();
    vec2 = execute_move_to_goal();
    /* you can multiply the two vectors by their gain values. */
    output = vec1 + vec2;
    /* combine the two vectors summation. */
    turn (output.direction);
    forward(output.magnitude* motor_schema.MAX_VOLTICITY);
}
```
APPENDIX B

MULTI-ROBOT SIMULATOR SOFTWARE

B.1 Teambots Simulation Environment

Several research institutions have released multi-robot simulation software. One of these simulation is Teambots [Balch 1997], was selected as a development platform for implementation our research ideas. Figure A.1 illustrate a sample Teambots simulation.

Figure B.1: The Teambots Simulation Environment.

Teambots, developed by Tacker Balch at Georgia Tech University and continued at Carnegie Mellon University, and is a Java-based collection of applications programs and Java packages for multi-agent mobile robotics research. Teambots supports prototyping, simulation and
Teambots was designed to accommodate the behavior-based control using *motor-schemas* [Arkin, 1998], which are defined with the flexible *Clay* toolkit. The default *Clay* Java packages provide a rich set of useful predefined schemas. Teambots permits the programmer to define new types of robots and new control systems by defining subclasses of existing abstract classes. Furthermore, Teambots Java-based class hierarchy is written so that the control system for one type of robot can be ported to another type of robot with little or no modification. Finally, Teambots parses a text description file to load the appropriate objects for simulation, including robots, obstacles, and targets. In simulation the robot kinematically holonomic vehicle (a simulated Nomadic Technologies’ Nomad 150) controlled by a behavioral system in *Clay*. Simulated motor and sensor capabilities are based on performance of physical robots. The robots can detect hazards with sonar out to a range of nine meters. Attractors can be detected visually out to here meter across a 90-degree field of view [Balch, 1998a].

![Figure B.2: The Teambots real robot.](image)
B.2 Foraging Task Behavioral Design

The robots implemented the schema-based behavior-based control methodology (cf. §2.4). In this approach, the robot is provided several pre-programmed collection of parallel, concurrently active behaviors, some of which gather sensor information, called perceptual-schema (also referred to as perceptual triggers) some derive effectors called motor-schema [Arkin, 1998]. The perceptual-schema generates binary perceptual features that represent the appropriate robot state space and are used to sequence the robot through steps in achieving the task. In foraging task, a robot has the following perceptual features,

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>at_home?</td>
<td>Check the state of the robot if it is at home or not.</td>
</tr>
<tr>
<td>target_visable?</td>
<td>Check the state of the robot if it encounters an attractor or not.</td>
</tr>
<tr>
<td>have_target?</td>
<td>Check the state of the robot if it grasps the attractor or not.</td>
</tr>
</tbody>
</table>

The robot must decide on the basis of these environmental cues which behavior to activate at each point in time. The appropriate motor-schema corresponding to actions in achieving the homogenous foraging task consists of the following behaviors:

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wander</td>
<td>Move randomly about the environment in search for attractors.</td>
</tr>
<tr>
<td>acquire</td>
<td>Move towards the closest visible attractor.</td>
</tr>
<tr>
<td>deliver</td>
<td>Move towards the delivery area.</td>
</tr>
</tbody>
</table>

Selection of the appropriate behavior, given the situation, may be hand coded or discovered by the robot through reinforcement learning. In this domain, robots select from only three behaviors based on eight situations.
APPENDIX C

BOUNDS WITH A LEARNING RATE $k$

Let $M$ be updated by:

$$M = (1 - k) \cdot M + k \cdot m$$

(C.1)

Where $m$ is bounded by $m_{\text{max}}$, $m_{\text{min}}$, and the initial value of $k = 1$. Then:

Theorem C.1: $M$ is also bonded by $m_{\text{max}}$, $m_{\text{min}}$.

Proof: The highest $M$ can be if it is always updated with $m_{\text{max}}$:

$$M = 0 \cdot M + 1 \cdot m_{\text{max}} = m_{\text{max}}$$

$$M = (1 - k) \cdot m_{\text{max}} + k \cdot m_{\text{max}} = m_{\text{max}}$$

(C.2)

... 

So $M_{\text{max}} = m_{\text{max}}$. Similarly $M_{\text{min}} = m_{\text{min}}$. 

$\blacksquare$
REFERENCES


137


141


ملخص الرسالة

يتزايد الاهتمام بالعمل الجماعي بين العديد من الإنسان الآلي لما له من تطبيقات متعددة في الحياة مثل مهام الفضاء، العمل في بيئات خطرة وتطبيقات عسكرية. بعض المهام التي تعلو لفريق من الإنسان الآلي تضم بعض الأنشطة التي يتم تنفيذها في نفس اللحظة. يحتاج العمل في فريق المزيد من التعاون بين أعضاء الفريق لأداء المهام المكلفة بها بكفاءة. تكتسب الفرق الكفاءة إذا استطاعت باستقلالية تعلم كيفية توزيع مواردها بين مختلف الأنشطة وتعديلها بتعاون حسب الطلب في تنفيذ المهمة. وسؤال الأسئلة كيف يتم هذا مع نظام متعدد الإنسان الآلي ذات التحكم اللازم؟ يدعى هذا القصور مشكلة التنظيم؟ وتحتاج هذا تعلم السلوك التنظيمي و التنظيمي. في هذا العمل البحثي تم تقديم وتطبيق واختبار خوارزمات جديدة للتعلم بالتعزيز لإكتساب خبرة التنظيم والتعاون في بيئة فريق الإنسان الآلي ولهد أدت هذه الخوارزمات إلى كفاءة في الأداء حيث مساعد في إنقاذ مجال البحث و الوقت لافضل خطة بفضل بعض المعلومات المساعدة.
كلية الهندسة
جامعة القاهرة

خوارزمات التعلم بالتعزيز وتطبيقها في تنظيمات متعددة الروبوت

إعداد
مهندس/ محمد السعيد إبراهيم التليائي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
للحصول على درجة دكتوراه في الفلسفة
في هندسة الإلكترونيات والإتصالات

تعتمد من لجنة الممتحنين:

المرشح الرئيسي
أ.د. سمير إبراهيم شاهين

مرشح
أ.د. أحمد حسن عبد الوهاب

عضوا
أ.د. محمد زكي عبد الجليل

عضوا
أ.د. أحمد إبراهيم إمام

الجيزه، جمهورية مصر العربية
يونيو 2003
كلية الهندسة
جامعة القاهرة

خوارزمات التعلم بالتعزيز وتطبيقها في تنظيمات متعددة الروبوتات

إعداد
المهندس/ محمد السيد إبراهيم التلبائي

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
للحصول على درجة دكتوراه الفلسفة
في هندسة الإلكترونيات والاتصالات

تحت إشراف
أ.د. أسير إبراهيم شاهين
قسم هندسة الخصائص
قسم الخصائص والنظم
معهد بحوث الإلكترونيات
كلية الهندسة- جامعة القاهرة

كلية الهندسة، جامعة القاهرة
الجيزة، جمهورية مصر العربية
يونيو 2003