Representation, Planning, and Learning of Dynamic Ad Hoc Robot Teams

Somchaya Liemhetcharat

CMU-RI-TR-13-22

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 2013

Thesis Committee:
Manuela M. Veloso, Chair
Howie Choset
M. Bernardine Dias
Peter Stone (University of Texas at Austin)

To my loving wife, Junyun Tay, for her endless support and encouragement.

Abstract

Forming an effective multi-robot team to perform a task is a key problem in many domains. The performance of a multi-robot team depends on the robots the team is composed of, where each robot has different capabilities. Team performance has previously been modeled as the sum of single-robot capabilities, and these capabilities are assumed to be known.

Is team performance just the sum of single-robot capabilities? This thesis is motivated by instances where agents perform differently depending on their teammates, i.e., there is *synergy* in the team. For example, in human sports teams, a well-trained team performs better than an all-stars team composed of top players from around the world. This thesis introduces a novel model of team synergy — the Synergy Graph model — where the performance of a team depends on each robot's individual capabilities and a task-based relationship among them.

Robots are capable of learning to collaborate and improving team performance over time, and this thesis explores how such robots are represented in the Synergy Graph Model. This thesis contributes a novel algorithm that allocates training instances for the robots to improve, so as to form an effective multi-robot team.

The goal of team formation is the optimal selection of a subset of robots to perform the task, and this thesis contributes team formation algorithms that use a Synergy Graph to form an effective multi-robot team with high performance. In particular, the performance of a team is modeled with a Normal distribution to represent the nondeterminism of the robots' actions in a dynamic world, and this thesis introduces the concept of a δ-*optimal team* that trades off risk versus reward. Further, robots may fail from time to time, and this thesis considers the formation of a robust multi-robot team that attains high performance even if failures occur.

This thesis considers *ad hoc teams*, where the robots of the team have not collaborated together, and so their capabilities and synergy are initially unknown. This thesis contributes a novel learning algorithm that uses observations of team performance to learn a Synergy Graph that models the capabilities and synergy of the team. Further, new robots may become available, and this thesis introduces an algorithm that iteratively updates a Synergy Graph with new robots.

This thesis validates the Synergy Graph model in extensive simulations and on real robots, such as the NAO humanoid robots, CreBots, and Lego Mindstorms NXTs. These robots vary in terms of their locomotion type, sensor capabilities, and processing power, and show that the Synergy Graph model is general and applicable to a wide range of robots. In the empirical evaluations, this thesis demonstrates the effectiveness of the Synergy Graph representation, planning, and learning in a rich spectrum of ad hoc team formation scenarios.

Acknowledgments

First and foremost, I would like to express my thanks and gratitude to my advisor, Manuela Veloso. She has provided invaluable guidance and support throughout my undergraduate and graduate life. Her creativity and energy have inspired me to greater heights.

I would also like to thank Howie Choset, Bernardine Dias, and Peter Stone, for graciously agreeing to be on my thesis committee, and devoting their precious time to provide invaluable guidance and advice for my thesis.

I am eternally grateful and blessed with my wife, Junyun Tay. She has been understanding and encouraging, and has dedicated much of her time supporting me during my Ph.D., discussing research questions and directions, analyzing my results, and carefully proof-reading my papers. Considering that she is also working on her Ph.D and taking care of our daughter at the same time, I am truly impressed and amazed at her capabilities!

My family has always been supportive, and I would like to thank my parents and parents-in-law for their encouragement, and concern for our well-being. Their love is truly boundless.

I am also thankful for my daughter, Dhanaphon, who motivated me to finish my thesis with her bright smiles, and inspired me with her incredible learning capabilities. I hope to someday develop artificial intelligence and learning algorithms that will rival a baby's abilities.

A Ph.D. is a long process, and I am thankful for the many friends I made along the way, such as the CORAL group at CMU, including Brian Coltin, Susana Brandao, and Çetin Meriçli, who I've had many fun experiences with. I would also like to thank Felipe Trevizan, who methodically read my thesis and offered many useful comments and suggestions, and Yucheng Low and Qirong Ho, who shared their knowledge of statistics and machine learning with me.

The Singaporean community in Pittsburgh have truly made living here feel like home, and my many friends, including Rodney and Suzanne, Jiaqi and Weiling, Chwee Ming and Andrea, Bryan and Cocoa, Kenneth and Linli, have been there through the ups and downs of life overseas, and I feel honored to be their friend.

I would also like to thank my friends, back in Singapore as well as overseas, such as Jun Wei Chuah, Winston Goh, Ramanpreet Singh, Rinawati Rahmat, Mike Phillips, Jonathan Wang, Junhao Soh, Dillon Ng, who have been in touch with me over the years and providing support.

I am also very grateful to the Agency for Science, Technology and Research (A*STAR) of Singapore, which graciously funded my undergraduate and graduate studies, and to Carnegie Mellon University, for allowing me to pursue my research interests here.

Lastly, I would like to thank the Reprisal guild of Steamwheedle Cartel in the World of Warcraft, notably Tom, Jess, Marco, Michele, and Brooks. We have spent many hours online saving the world (of Warcraft) together, and our friendships have brought me much joy and laughter.

Contents

1 Introduction **1**

1.1 Thesis Question and Approach . . . 3

1.1.1 Representing Capabilities and Synergy . . . 3

1.1.2 Planning Effective Multi-Robot Teams . . . 5

1.1.3 Learning the Capabilities and Synergy . . . 6

1.1.4 Evaluation . . . 7

1.2 Thesis Contributions . . . 9

1.3 Document Outline . . . 10

2 Synergy Graph Model and Team Formation **13**

2.1 Team Formation Problem . . . 13

2.2 Task-Based Relationships . . . 14

2.3 Agent Capabilities . . . 16

2.4 Unweighted Synergy Graph Model . . . 17

2.5 Weighted Synergy Graph Model . . . 19

2.5.1 A Weighted Synergy Graph Example . . . 20

2.5.2 Equivalence in Weighted Synergy Graphs . . . 21

2.6 Assumptions of the Synergy Graph Model . . . 22

2.7 Solving the Team Formation Problem . . . 23

2.7.1 Forming the δ-Optimal Team . . . 24

2.7.2 Approximating the δ-Optimal Team . . . 26

2.7.3 Comparing the Team Formation Algorithms . . . 27

2.8 Comparing Unweighted and Weighted Synergy Graphs . . . 29

2.8.1 Experimental Setup . . . 29

2.8.2 Comparison Results . . . 30

2.9 Chapter Summary . . . 32

3 Learning Synergy Graphs **33**
3.1 Overview of the Learning Algorithm . . . 33
3.2 Learning the Synergy Graph Structure . . . 35
3.2.1 Generating a Random Synergy Graph Structure . . . 36
3.2.2 Generating a Neighbor Synergy Graph Structure . . . 36
3.3 Learning Capabilities . . . 38
3.3.1 Learning Capabilities with a Least-Squares Solver . . . 38
3.3.2 Learning Capabilities with a Non-Linear Solver . . . 41
3.4 Computing Log-Likelihood and Accepting Neighbors . . . 41
3.5 Evaluating the Learning Algorithm . . . 42
3.5.1 Learning Unweighted Synergy Graphs . . . 42
3.5.2 Learning Representative Weighted Graph Structures . . . 45
3.5.3 Learning Random Weighted Synergy Graphs . . . 47
3.5.4 Comparing the Capability Learning Algorithms . . . 49
3.6 Chapter Summary . . . 50

4 Iteratively Learning a New Teammate **51**
4.1 Learning Algorithm for Adding a Teammate . . . 51
4.2 Generating the Teammate's Initial Edges . . . 54
4.3 Generating Neighbor Edges . . . 57
4.4 Learning the Teammate's Capability . . . 57
4.5 Analyzing the Iterative Learning Algorithm . . . 58
4.5.1 Experimental Setup . . . 58
4.5.2 Comparison Results . . . 60
4.5.3 Comparing Different Learning Approaches . . . 62
4.6 Chapter Summary . . . 63

5 Modifications to the Synergy Graph Model **65**
5.1 Agents with Multiple Capabilities . . . 65
5.1.1 Role Assignment Problem Definition . . . 66
5.1.2 Weighted Synergy Graph for Role Assignment (WeSGRA) Model . . . 67
5.1.3 Finding Role Assignments and Learning WeSGRAs . . . 69
5.1.4 Experiments with WeSGRAs . . . 70
5.2 Graphs with Multi-Edges . . . 72
5.2.1 Configurable Team Formation Problem Definition . . . 73
5.2.2 Synergy Graph for Configurable Robots (SGraCR) Model . . . 75

5.2.3 Configuring Multi-Robot Teams and Learning SGraCRs 78
5.3 Non-transitive Task-Based Relationships 79
5.3.1 Modeling Non-Transitivity in Synergy Graphs 79
5.3.2 Implications of Non-Transitive Synergy 80
5.4 Chapter Summary 82

6 Agents with Complex Characteristics 83
6.1 Agents that Probabilistically Fail 83
6.1.1 Robust Team Formation Problem Definition 83
6.1.2 Robust Synergy Graph for Configurable Robots (ρ-SGraCR) Model . . . 87
6.1.3 Solving the Robust Team Formation Problem 88
6.1.4 Evaluating the ρ-SGraCR Model 89
6.1.5 Comparing the Robust Team Formation Algorithms 90
6.2 Agents that Learn to Coordinate Better over Time 91
6.2.1 Dynamic Weighted Synergy Graph (DyWeSG) Model 92
6.2.2 Solving the Learning Agents Problem 95
6.2.3 Evaluating the Algorithms 98
6.3 Chapter Summary 101

7 Applications and Results 103
7.1 Team Formation with Probabilistic Robot Capabilities 103
7.1.1 Probabilistic Model of Robot Capabilities 104
7.1.2 Experimental Setup 104
7.1.3 Experimental Results 106
7.2 Team Formation in RoboCup Rescue 106
7.2.1 The RoboCup Rescue Simulator 107
7.2.2 Experimental Setup 108
7.2.3 Experimental Results 110
7.3 Role Assignment in RoboCup Rescue 111
7.3.1 Experimental Setup 111
7.3.2 Experimental Results 112
7.4 Role Assignment in a Foraging Task 113
7.4.1 The Foraging Task 113
7.4.2 Experimental Setup 114
7.4.3 Experimental Results 115
7.5 Configuring a Team for a Manufacturing Task 116

7.5.1 Experimental Setup . . . 116
7.5.2 Experiments with Synthetic Data . . . 117
7.5.3 Experiments with Simulated Robots . . . 118
7.5.4 Experiments with Real Robots . . . 118
7.6 Robust Team Formation in a Foraging Task . . . 120
7.6.1 The Foraging Task . . . 120
7.6.2 Robot Types and Behaviors . . . 122
7.6.3 Experimental Setup . . . 123
7.6.4 Experimental Results . . . 124
7.7 Chapter Summary . . . 125

8 Related Work **127**
8.1 Multi-Robot Task Allocation . . . 127
8.2 Role Assignment . . . 130
8.3 Coalition Formation . . . 131
8.4 Ad Hoc Teams . . . 135
8.5 Team Formation . . . 136
8.6 Operations Research . . . 136
8.7 Robustness and Redundancy . . . 137

9 Conclusion and Future Work **139**
9.1 Contributions . . . 139
9.2 Bridging Previous Work and Future Work . . . 141
9.3 Future Directions . . . 145
9.4 Concluding Remarks . . . 147

Bibliography **149**

List of Figures

1.1 Three robot platforms used to evaluate this thesis. a) Aldebaran NAO humanoid robot; b) CreBot: our version of the TurtleBot; c) Lego Mindstorms NXT. 8

1.2 Overview of the thesis approach, and organization of the chapters. 11

2.1 a) A fully-connected task-based graph with 4 agents, where the edge weights represent the cost of agents working together to perform the task. b) A connected task-based graph with the same 4 agents, where some edges have been removed while still preserving the pairwise distances between agents. 15

2.2 Task-based graphs with 3 agents, where the agents' heterogeneous capabilities are attached to each vertex and are represented as a) values; b) Normally-distributed variables. 17

2.3 An Unweighted Synergy Graph with 5 agents. Each vertex represents an agent, and the distance between vertices in the graph indicate how well agents work together. Agent capabilities are modeled as Normally-distributed variables. . . . 19

2.4 a) A weighted task-based graph with 3 agents. b) An unweighted task-based graph with the 3 agents (all edges have a weight of 1). If the compatibility function in (a) is $\phi(d) = \frac{1}{d}$, the task-based relationships of the agents cannot be represented with an unweighted graph and an adjusted compatibility function. . . 20

2.5 An example of a Weighted Synergy Graph modeling capabilities and the task-based relationships of a group of agents in a rescue task. 21

2.6 Three equivalent Weighted Synergy Graphs, i.e., the shortest distance between pairs of agents is equivalent in the three graphs. 22

2.7 Effectiveness of teams found in the learned Weighted Synergy Graph and learned Unweighted Synergy Graph, using simulated annealing with 1000 iterations. The compatibility function was $\phi_{\text{fraction}}(d) = \frac{1}{d}$. 31

2.8 Average effectiveness of teams found in the learned Weighted and Unweighted Synergy Graphs, using 1000 and 2000 iterations of simulated annealing to learn the Synergy Graph structure, using $\phi_{\text{fraction}}(d) = \frac{1}{d}$. 31

2.9 Effectiveness of teams found in the learned Weighted and Unweighted Synergy Graphs with $\phi_{\text{decay}}(d) = \exp\left(-\frac{d \ln 2}{2}\right)$, and 1000 iterations of simulated annealing. 31

3.1 The process of learning from observations. The individual capabilities of agents in the Synergy Graphs are not shown. 34

3.2 The four possible actions used to generate neighbor Weighted Synergy Graph structures. 37

3.3 The capabilities of agents are learned from the observation set and a Weighted Synergy Graph structure using a least-squares solver. 40

3.4 The error in the learned Unweighted Synergy Graph with varying number of agents and both compatibility functions. 44

3.5 The error in the learned Unweighted Synergy Graph of 10 agents with heterogeneous task performance, using the compatibility function a) $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{3})$ and b) $\phi_{\text{fraction}}(d) = \frac{1}{d}$, compared with the initial Unweighted Synergy Graph used by the learning algorithm, with random structure but learned agent capabilities. 44

3.6 Results of the learning algorithm using representative graph structure types. The agent capabilities are not shown, and the vertices are laid out for visual purposes. a) Examples of Weighted Synergy Graphs with 5 agents generated to form representative structure types. b) The initial randomly-generated Weighted Synergy Graph of the learning algorithm. c) Learned Weighted Synergy Graphs corresponding to the Weighted Synergy Graphs in *(a)*, after 1000 iterations of simulated annealing. 46

3.7 Performance of the learning algorithm with different Weighted Synergy Graph structure types, averaged over 100 trials. 47

3.8 Performance of the learning algorithm on random weighted graph structures with varying number of agents, averaged over 100 trials. 48

3.9 Learning curves of two compatibility functions — ϕ_{decay} and ϕ_{fraction}, averaged over 100 trials. 48

3.10 Log-likelihoods of learned Weighted Synergy Graphs using a least-squares solver and non-linear solver to learn the agent capabilities, compared to the log-likelihood of the hidden Weighted Synergy Graph. 50

4.1 The learning algorithm `AddTeammateToSynergyGraph` adds a new teammate into a Synergy Graph (a Weighted Synergy Graph is used in this figure). Simulated annealing is performed, where edges of a_{N+1} are modified, and the capability C_{N+1} is learned. 52

4.2 The experimental process to compare the initial edge generation functions and capability learning methods. 59

4.3 Examples of the four Weighted Synergy Graph structure types generated: chain, loop, star, and random. 59

5.1 An example of the distances among three agent types $a_1, a_2, a_3 \in \mathcal{A}$, where a low distance between agent types reflects high compatibility and vice versa. . . . 68

5.2 The experimental process to evaluate the learning and team formation algorithms for WeSGRA. 71

5.3 Learning curve of the learning algorithm using training examples generated by a hidden WeSGRA model. 71

5.4 A SGraCR with 6 vertices, modeling two types of modules (shown in different shades). The edges with two weights indicate the intra and inter-robot weights respectively, and the self-looping edges have inter-robot weights. 77

5.5 A Weighted Synergy Graph with four agents. When the task-based relationship is assumed to be transitive, the edge $\{a_1, a_2\}$ is never used in the computation of synergy. 80

5.6 Two non-transitive Synergy Graphs where the team $\{a_1, a_2, a_3\}$ is valid. 82

6.1 A Dynamic Weighted Synergy Graph with 5 agents. Agent pairs that learn to coordinate better over time are denoted with bold edges. Initial edge weights are shown in black text, and the learning rates are shown in blue. 94

6.2 The performance of teams formed after K training instances by various heuristics. The dotted black line shows the performance of the optimal team. 100

7.1 An example of probabilistic robot capabilities. The numbers indicate probabilities of success, and the dashed lines out of actions 2 and 3 indicate that both are required to trigger the desired output. 104

7.2 The experimental process to compare our Synergy Graph algorithms against the ASyMTRe algorithm. 105

7.3 Screenshot of the RoboCup Rescue simulator showing the initial positions of the simulated robots. Green, red, blue, and white circles are civilians, fire engines, police cars, and ambulances respectively. The grey areas indicate buildings that darken as they burn down. 107
7.4 Learning curve of the Synergy Graph learning algorithm using cross-validation of data from the RoboCup Rescue simulator. 112
7.5 The distribution of values of role assignment policies. The values in the training examples is shown as a cross (the mean) with horizontal lines showing the standard deviation. 113
7.6 The experimental setup for the foraging experiments. The red circle indicates a hidden fifth ball, and the blue circle indicates where balls are replaced if they are moved past the side and back lines. Different combinations of robots were placed in the 3 robot roles $\mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3$. 114
7.7 a) The layout of the experiments involving NXT robots transporting items from L_0 to L_3. b) A NXT robot as it approaches L_1. 119
7.8 The three robot platforms used in the foraging task: a) Lego NXT; b) CreBot; c) Aldebaran NAO. 121
7.9 The setup of the foraging experiment showing the initial robot positions and wooden block positions. Uncolored and colored wooden blocks are to be foraged to their respective stockpiles on the left and right sides of the field. 121
7.10 The robustness scores of teams formed by ρ-SGraCR and competing approaches. The dark blue line indicates the median, the top and bottom of the box represent the 75^{th} and 25^{th} percentiles, and the top and bottom whiskers represent the maximum and minimum values. 124

List of Tables

2.1 The number of evaluations done by the algorithms `Form`δ`OptimalTeam` and `Approx`δ`OptimalTeam` to compute and approximate the δ-optimal team respectively in a Weighted Synergy Graph, the time taken by the algorithms in milliseconds, and the quality of the team found (where 0 means the worst team and 1 is the optimal team). 28

2.2 The number of evaluations done by the algorithms `Form`δ`OptimalTeam` and `Approx`δ`OptimalTeam` in an Unweighted Synergy Graph, the time taken in milliseconds, and the effectiveness of the team found. 28

3.1 Effectiveness of multi-agent teams formed. 45

3.2 Scaled log-likelihood of the learned Weighted Synergy Graphs, varying the number of agents, and γ, the scale-factor of agent capabilities. 48

4.1 Average difference $D(S^*, S^+)$ between the hidden and learned Weighted Synergy Graphs given different hidden structure types and capability learning algorithms. 61

4.2 Average difference between the hidden and learned Weighted Synergy Graphs using different learning methods. 63

6.1 The optimal robustness scores of teams with 1 robot (3 modules) to 4 robots (12 modules). 90

6.2 The robustness scores of teams formed by `ApproxOptimalRobustTeam` compared to the optimal team formed by `FormOptimalRobustTeam`. 91

6.3 Regret of various heuristics versus the optimal allocation of training instances. . . 101

7.1 Effectiveness of teams formed by using the Unweighted Synergy Graph versus ASyMTRe. 106

7.2 Average scores of combinations of algorithms in the RoboCup Rescue simulator, formed by the Weighted Synergy Graph model, Unweighted Synergy Graph model, and IQ-ASyMTRe. 110
7.3 Effectiveness of algorithms in the foraging domain with real robots. 116
7.4 Experimental results of SGraCR and two competing approaches using synthetic data derived from a hidden SGraCR model, simulated robots in a manufacturing scenario, and robot experiments using Lego NXT robots. The scores indicate the number of standard deviations above the mean, i.e., a score of x means that the approach found a team with a value $\mu + x\sigma$, where μ and σ are the mean and standard deviation of values of teams. 117

Chapter 1

Introduction

The performance of a multi-robot[1] team depends on the robots the team is composed of. The problem of team formation is the selection of the optimal subset of robots from a larger set, where the optimal team has the highest performance. Team formation has similarities with multi-robot task allocation (MRTA) [Gerkey and Mataric, 2004] and coalition formation [Sandholm et al., 1999], with the key difference that only a single team with highest performance is formed in team formation, while MRTA optimizes the sum of utilities from completed tasks, and coalition formation optimizes the sum of utilities of coalitions. Research in robot capabilities has focused on single-robot capabilities, e.g., the carrying capacity of a robot, and the performance of a team is the sum of single-robot capabilities [Shehory and Kraus, 1998].

We understand that there is *synergy* among the robots in a team, where team performance at a particular task depends not only on the robots' individual capabilities, but also on the composition of the team itself. There are many illustrations of synergy in real human teams, basically for any task. An example is an all-star sports team comprised of top players from around the world, hence individual agents with high capabilities, who may have a lower synergy as a team and perform worse than a well-trained team of individuals with lower capabilities but much higher synergy. We consider a "team" as a group of robots that coordinate together to perform a task, as compared to a general multi-robot system where the robots may perform tasks independently. In particular, we are interested in problems for which a representation of team performance that goes beyond the sum of single-robot capabilities is needed. Specific robots may have or acquire a high task-based relationship that allows them to perform better as a team than other robots with equivalent individual capabilities but a low task-based relationship. This thesis introduces a novel model of team performance that incorporates individual capabilities and team synergy.

We are motivated by research in ad hoc agents that learn to collaborate with previously un-

[1]This thesis applies to robots and software agents. Hence, we use the terms *robot* and *agent* interchangeably.

known teammates [Stone et al., 2010]. This thesis investigates the impact of learning agents, and contributes an algorithm that iteratively selects opportunities for the learning agents to improve their performance, so as to form the highest-performing multi-robot team at the end of a fixed number of iterations. In particular, we model pairs of agents whose team performance improves over time. Hence, by focusing on team formation with learning agents, this thesis complements the growing research on ad hoc agents and other learning agents.

When robots act in a dynamic environment, their performance may be nondeterministic, e.g., wheel slippage of a robot may cause it to take slightly different times to reach a location. We represent capabilities and team performance as random variables to capture the nondeterminism, which is novel compared to representing performance as a utility value. However, when team performance is represented as a random variable, the definition of the optimal team has to account for the nondeterminism. We introduce a notion of optimality that ranks teams based on the probabilistic performance, and provides a measure of risk versus reward.

Further, robots may occasionally fail and be unable to perform a task. This thesis also investigates team robustness and defines the optimal robust team. Two team formation algorithms are introduced to form and approximate the optimal team assuming no failures occur, and two robust team formation algorithms are introduced to form multi-robot teams that are robust to failures of their members. We are interested in solutions to the team formation problem that are computationally feasible, and hence we contribute approximation algorithms that efficiently converge to near-optimal solutions.

Most existing team formation approaches assume that the robot capabilities are known *a priori* (e.g., [Zhang and Parker, 2012]). An ad hoc team is one that is formed for a particular task, where the robots in the team may not have collaborated with each other. Assuming an ad hoc team, we address the team synergy learning question as: given a set of robots with unknown capabilities, how do we model and learn the capabilities and synergy of the robots through observations, in order to form an effective team, i.e., a subset of the robots? A solution to this problem will enable ad hoc teams to be applied to a variety of problems in the real world, where effective teams will inevitably need to be composed of robots who may not have previously worked together, or have not been developed by the same team of researchers.

Hence, we want to learn team synergy from data. We assume that some observations of the performance of teams are given (similar to tryouts on human teams), and the learning goal is to fit the best model to the training data. By using only observations of team performance for learning, and not requiring additional domain information, we are able to apply our model and algorithms to a wide range of multi-robot problems. For example, we are motivated by the urban search-and-rescue domain, where researchers from around the world have developed

various rescue robots and algorithms. This thesis seeks to learn the capabilities and synergy of such robots and algorithms, in order to form an ad hoc team when a disaster strikes. The thesis is not limited to any particular domain, and we demonstrate our model and algorithms in a variety of real robot scenarios.

Sometimes not all data is available upfront. For example, a new robot may join the set of available robots but there is no prior information about it. This thesis introduces two learning algorithms: one that learns the model from a set of observations, and one that incorporates a new robot into an existing model. The second learning algorithm, which incorporates a new robot into the existing model, can be run iteratively to update the model as new robots become available, thus alleviating the need to completely relearn the model whenever new information is received.

Thus, this thesis investigates how to represent the capabilities and synergy of robots, how to form an effective multi-robot team, and how to learn the model of capabilities and synergy from observations. This thesis evaluates the Synergy Graph model and algorithms in a variety of simulated and real robot experiments, in domains such as urban search-and-rescue and foraging. We demonstrate that our approach is applicable to a wide range of multi-robot problems, and forms effective teams that outperform competing approaches.

1.1 Thesis Question and Approach

The thesis question is:

Given a set of agents with non-deterministic performance at a task, how:

- **to model synergistic effects among members of an ad hoc team;**
- **to learn such a model through observations of cooperative robots; and**
- **to form an effective team?**

We present our approach to the thesis question by further dividing it into three categories: representation; planning; and learning, with two sub-questions per category.

1.1.1 Representing Capabilities and Synergy

How can the capabilities of heterogeneous robots and their synergy in a multi-robot team be represented?

We contribute a representation of team performance that goes beyond the sum of single-robot capabilities; there is a notion of *synergy* among the robots in the team, where team performance depends not only on the robots' individual capabilities, but also a task-based relationship among

them. To model the task-based relationship, this thesis introduces a graph structure where the robots are vertices in the graph, and edges represent the task-based relationship. In such graphs, we define the level of synergy of a set of robots as a function of the shortest path between them. We further devise a non-binary metric of team performance based on a Gaussian model of the individual robot capabilities. Such probabilistic variables capture the inherent variability in team performance in a dynamic world.

This thesis formally defines the Synergy Graph model, and how it is used to compute the synergy of a multi-robot team. We introduce the Unweighted Synergy Graph model, where the edges in the graph are unweighted (i.e., unit distance), and the Weighted Synergy Graph model that has weighted edges in the graph, and compare the two models. The Synergy Graph model is general and easily modified for specialized needs.

This thesis presents modifications to the general Synergy Graph model that allow it to be used in a wide variety of problems. First, by considering robots with multiple capabilities and self-loops in the graph structure, the Synergy Graph is applied to the role assignment domain. Second, by considering graphs with multi-edges, specifically two edges between every pair of vertices, and self-loops between each vertex, we apply the Synergy Graph model to configurable multi-robot teams where each robot is a selection of relevant robot modules. Instead of having a vertex represent a single robot (as in the Unweighted and Weighted Synergy Graph models), we consider using each vertex to represent a module of a robot. The multi-edges between module vertices model the synergy of modules within a robot as well as across robots. Third, the Synergy Graph model assumes that the task-based relationship is transitive (the shortest distance between robots is always used). We consider non-transitive task-based relationships, and explore the expressiveness of the updated model and its implications on the Synergy Graph algorithms.

How to model robots that learn to improve coordination?

We investigate how the Synergy Graph model can represent robots that learn from experience to improve their coordination in a team, i.e., their performance improves as a function of the number of learning instances they have had. In particular, we are inspired by research in ad hoc agents [Stone et al., 2010], that learn and model their teammates in order to improve their coordination and team performance.

This thesis models such improvements in coordination as changes in the edge weights of the Synergy Graph structure. In addition to modeling learning robots, this thesis further investigates the associated learning and team formation problem, i.e., given a fixed number of instances to learn and the goal of forming a multi-robot team at the end of the instances, how do we trade off between exploration (improving our estimate of the learning rate of the robots) and exploitation

(improving the final team score)? The learning robots problem has many similarities with the multi-armed bandit problem, and this thesis explores these in detail. We highlight important differences between the two problems, and evaluate the performance of heuristics from the bandit problem when they are applied to the learning robots problem.

1.1.2 Planning Effective Multi-Robot Teams

In a dynamic environment where the performance of robot teams is non-deterministic, how is the optimal multi-robot team formed?

The Synergy Graph model uses Normally-distributed variables to model robot capabilities and team performance. This thesis introduces the concept of δ-optimality, that ranks teams based on a probabilistic measure, i.e., the δ-optimal team is one that has the highest value with probability δ. The probability δ determines whether the δ-optimal team is risky or risk-adverse. When $\delta = \frac{1}{2}$, only the mean performance is considered, and the δ-optimal team is equivalent to the standard optimal team (commonly used by other approaches) that uses utility values. When $\delta < \frac{1}{2}$, a risky team is preferred, where there is a low probability of attaining a high value (high risk, high reward). Conversely, when $\delta > \frac{1}{2}$, a risk-adverse team is preferred that has a high probability of attaining a low value (low risk, low reward).

This thesis uses the concept of δ-optimality and contributes two team formation algorithms, `Form`δ`OptimalTeam` and `Approx`δ`OptimalTeam`. The inputs to both algorithms are a Synergy Graph, and the number of robots in the δ-optimal team. When the number of robots is unknown, the algorithms are run iteratively over all possible sizes. The first algorithm, `Form`δ`OptimalTeam`, uses branch-and-bound to form the δ-optimal team, by estimating the minimum and maximum team performance bounds of possible teams. `Form`δ`OptimalTeam` runs in $\mathcal{O}(N^n)$, where N is the total number of robots, and n is the number of robots in the team. The second algorithm, `Approx`δ`OptimalTeam`, uses an approximation algorithm to explore the space of possible teams, and approximates the δ-optimal team in $\mathcal{O}(n^2)$. This thesis compares the effectiveness of teams found by the two algorithms against the amount of search space explored.

How do potential failures in robots affect optimality, and how is the optimal robust multi-robot team formed?

Robots have variable performance due to the dynamics of the world, e.g., errors in movement from wheel slippage, or from inaccurate localization. Robots may also completely fail to function, e.g., a breakdown of a key component, or a battery running out of power. This thesis investigates how such failures are modeled and accounted for. We detail how the optimal robust team is defined in two ways: i.e., the team that attains the highest value with probability δ taking into account potential failures, and the team that maximizes the probability of attaining a minimum value.

The two measures of robust optimality are closely-linked, and we contribute two robust team formation algorithms that solve the first robust optimality measure (the algorithms can be modified for the second optimality measure). `FormOptimalRobustTeam` iterates through all possible teams and computes the optimal robust team in exponential time, by considering all possible failures in the multi-robot team. The second robust team formation algorithm, `ApproxOptimalRobustTeam`, approximates the optimal robust team by making assumptions in the failures of the robots in the teams and runs in polynomial time, and is more scalable to real-world problems.

1.1.3 Learning the Capabilities and Synergy

How can the model of robot capabilities and synergy be learned using observations of team performance?

This thesis presents `LearnSynergyGraph`, an algorithm that learns a Synergy Graph using only observations of team performance. As such, the learning and team formation algorithms enable the Synergy Graph model to be applied to a variety of problems, since the only input is the training observations and no additional information about the problem domain is required.

The learning algorithm learns the structure of the Synergy Graph by iteratively improving the graph structure, and learns the robots' capabilities using the observations and the graph structure. The Synergy Graph structure is iteratively modified by randomly selecting discrete actions on the graph structure, such as adding and removing edges. Using the graph structure and synergy equations, the observations of team performance are used to learn the robot's capabilities. This thesis presents two methods of learning robot capabilities, by using a least-squares solver and a non-linear solver. We show that the learning algorithm learns representative graph structure types and random Synergy Graphs effectively, and that the learned Synergy Graph is used to form effective multi-robot teams with the team formation algorithms.

How can information about a new robot be incorporated into a learned model?

The learning algorithm `LearnSynergyGraph` assumes that observations of all the robots are available initially, and learns the best Synergy Graph that fits the observations. However, not all observations may be readily available. Specifically, information about new robots may only be available at a later time. This thesis presents `AddTeammateToSynergyGraph`, an algorithm that uses observations of a new teammate to incorporate it into an existing Synergy Graph. The algorithm creates a new vertex to represent the new robot, and edges that connect the new vertex to the existing vertices in the Synergy Graph. We explore three heuristics to create the initial edges of the new vertex, and experimentally compare the effectiveness of these heuristics in a variety of Synergy Graph structures.

`LearnSynergyGraph` can also be used to incorporate information about a new teammate, namely by relearning the entire Synergy Graph using the union of the old and new observations. This thesis compares three learning approaches: completely relearning the Synergy Graph using `LearnSynergyGraph`; learning the Synergy Graph using the old observations with `LearnSynergyGraph` and adding a teammate with `AddTeammateToSynergyGraph`; iteratively adding all the robots with `AddTeammateToSynergyGraph`.

1.1.4 Evaluation

We evaluate the model and algorithms presented in this thesis in several simulated and real robot experimental domains. A complete description of the domains and experimental process is included with each evaluation.

Figure 1.1 shows the three platforms we use in this thesis: Aldebaran NAO humanoid robot, CreBot, and Lego Mindstorms NXT. These robot platforms vary in terms of:

- computational power: the NAOs and CreBots have full-fledged PCs, while the NXTs have microcontrollers;
- configurability: the hardware on the NAOs are fixed, the NXTs are completely configurable, and the CreBots are in between;
- sensors: the NAOs use video cameras, the CreBots use Kinects, and the NXTs use light sensors;
- mobility: the NAOs walk on two legs, while the CreBots and NXTs use wheels;

Thus, the three robot platforms we chose for this thesis span a wide range of characteristics. They were chosen to demonstrate that this thesis is applicable to general robots, since our approach does not require any specialized knowledge of the platforms.

(a) (b) (c)

Figure 1.1: Three robot platforms used to evaluate this thesis. a) Aldebaran NAO humanoid robot; b) CreBot: our version of the TurtleBot; c) Lego Mindstorms NXT.

The Aldebaran NAO humanoid robot is produced by Aldebaran Robotics, with updates to the hardware approximately every year. In this thesis, we use the RoboCup edition NAO robot (H21). The NAO is a fully-autonomous robot with an internal CPU and wireless capabilities. I began my work with the Aldebaran NAO in 2008 through the RoboCup Standard Platform League (SPL) [RoboCupSPL, 2013], where two teams of autonomous NAOs attempt to score goals against each other in robot soccer. Through the RoboCup SPL, I realized that small differences in the manufacturing of the NAOs (they were assembled by hand by engineers at Aldebaran Robotics) and varying wear-and-tear on the motors cause differences in performance during robot soccer. In particular, we would manually pick robots for specific roles, such as using a less worn-out robot as the main attacker since it would need to walk quickly and stably, and a more worn-out robot for the goalkeeper as it was stationary for most of the game. From my experience in RoboCup, I decided to pursue a formal and quantitative approach to the team formation problem for my thesis.

The iRobot Create robot is commercially available from iRobot, and is a modification of the Roomba robotic vacuum system — the Create does not contain the vacuum and is more easily programmed. In this thesis, we use the CreBot infrastructure, that uses the Willow Garage TurtleBot hardware and software for low-level control, and the CoBot infrastructure for high-level control. I first used the iRobot Create platform for the LANdroids research project, where the goal was to autonomously establish a connected wireless network in unknown indoor environments. The CoBot project was also on-going in the CORAL research group that I am a part of (although I did not work directly on the CoBot project), with the goal of creating an autonomous robot in an office environment that would perform tasks such as delivering items from office to office. The CoBot infrastructure was written in the Robot Operating System (ROS) and was compatible with the TurtleBot hardware and software. Hence, the CreBot platform was available for the experiments of this thesis[2].

[2]We thank Brian Coltin and Joydeep Biswas for their work in developing the CreBot platform.

The Lego Mindstorms NXT is commercially available as a kit containing the NXT Intelligent Brick, and various Lego connectors, sensors and motors. We use an additional XBee radio that allows the NXTs to communicate wirelessly. I had prior experience with the Lego Mindstorms RCX (an older version of Mindstorms), and used the NXTs as part of the FIRE research project that involved developing curriculum in multi-robot communication. My experience with the NXTs in the FIRE project inspired me to consider configurable robots as a part of the thesis, as robots built with the NXT kit are easily redesigned and reconfigured.

1.2 Thesis Contributions

The contributions of this thesis are:

- Representation
  - The Synergy Graph model, where the performance of a multi-robot team depends on the robots' capabilities and the task-based relationship among them;
  - Extensions to the Synergy Graph model for applications in the role assignment domain, configuring robot modules, capturing the potential failures of robots, and modeling complex task-based relationships among the robots;
  - A representation of robots that learn to collaborate better over time, and an algorithm that iteratively learns the rate of improvement of the learning robots, in order to form the optimal team at the end of a fixed number of iterations;
- Planning
  - Two team formation algorithms that form and approximate the optimal multi-robot team given a Synergy Graph, assuming there are no failures in the robots;
  - Two robust team formation algorithms that form and approximate the optimal robust multi-robot team, taking possible failures into account;
- Learning
  - The Synergy Graph learning algorithm that learns a synergy graph from observations by iteratively improving the Synergy Graph structure and learning the robots' capabilities;
  - A learning algorithm that learns the capabilities and synergy of a new teammate and adds it into an existing Synergy Graph;
- Evaluation of the Synergy Graph model and algorithms in simulated and real robot domains, with Aldebaran NAO, CreBot, and Lego Mindstorms NXT robots.

1.3 Document Outline

Figure 1.2 presents an overview of the thesis approach and organization of the chapters. The outline below presents a summary of the chapters that follow:

Chapter 2 – Synergy Graph Model and Team Formation. We define the team formation problem and the δ-optimal team, and present the Unweighted and Weighted Synergy Graph models and how they model robot capabilities and task-based relationships. We present two team formation algorithms that form and approximate the δ-optimal team respectively.

Chapter 3 – Learning Synergy Graphs. We present a learning algorithm that learns the Synergy Graph model from data. We evaluate the learning algorithm using synthetic data, and show that effective teams are formed using the learned Synergy Graphs.

Chapter 4 – Iteratively Learning a New Teammate. We present an algorithm that uses observations of a new teammate to incorporate it into an existing Synergy Graph. We evaluate three heuristics that initialize the algorithm, and compare different Synergy Graph learning approaches given new observations.

Chapter 5 – Modifications to the Synergy Graph Model. We present modifications to the Synergy Graph model, such as agents with multiple capabilities, graphs with multi-edges, and non-transitive task-based relationships. These modifications allows Synergy Graphs to be applied to other problems such as role assignment and forming a multi-robot team by selecting modules of configurable robots.

Chapter 6 – Agents with Complex Characteristics. We present how agents with complex characteristics are represented in the Synergy Graph model, such as agents that probabilistically fail, and agents that learn over time. We formally define a robust team, and present two algorithms that forms and approximates the optimal robust team respectively. We discuss learning agents and how they are represented in the Synergy Graph model. We present an algorithm that iteratively learns the rate of improvement of the learning agents, in order to form the optimal team at the end of a fixed number of iterations.

Chapter 7 – Applications and Results. We present the application of the Synergy Graph model and algorithms in a variety of simulated and real robot domains, such as urban search-and-rescue and foraging. We demonstrate that our approach forms near-optimal teams and outperforms competing algorithms.

Chapter 8 – Related Work. We discuss previous work in areas such as: multi-robot task allocation; coalition formation; and team formation, and we discuss how they relate to this thesis.

Chapter 9 – Conclusion and Future Work. We conclude the dissertation with a summary of its contributions along with a discussion of promising directions for future work.

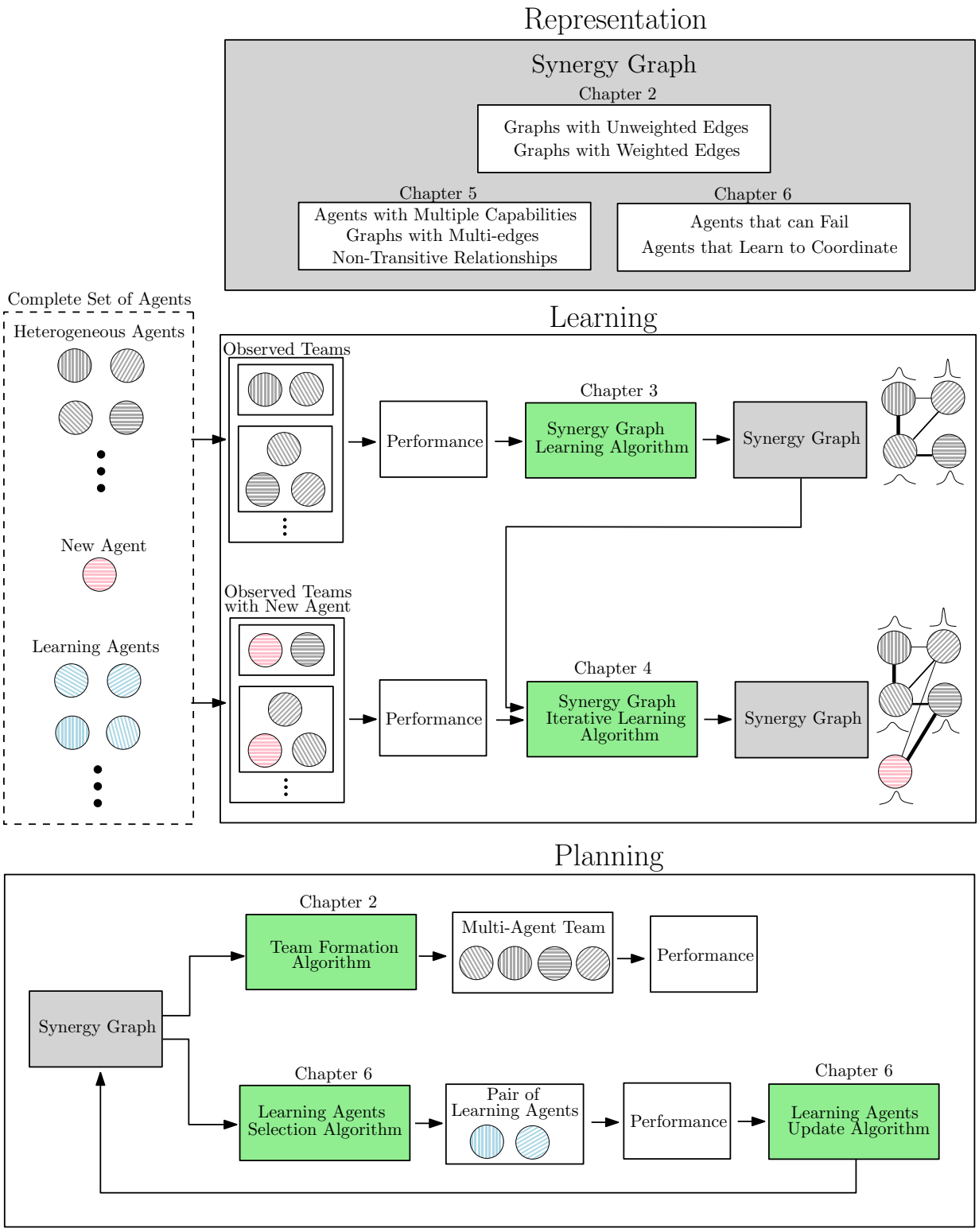


Figure 1.2: Overview of the thesis approach, and organization of the chapters.

Chapter 2

Synergy Graph Model and Team Formation

This chapter presents the Synergy Graph model and how it models robot capabilities and task-based relationships among the robots [Liemhetcharat and Veloso, 2012a, Liemhetcharat and Veloso, 2013d]. We begin with a formal definition of the team formation problem that is used throughout this document.

2.1 Team Formation Problem

We begin with the definition of the set of agents and the definition of a team:

Definition 2.1.1. *The **set of agents** is $\mathcal{A} = \{a_1, \ldots, a_N\}$, where each $a_n \in \mathcal{A}$ is an agent.*

Definition 2.1.2. *A **team** is any subset $A \subseteq \mathcal{A}$.*

There is a task to be performed that can be accomplished with *any* number of agents with varying performance, and hence any subset of $\mathcal{A}$ is a valid team. The performance of a team is the utility attained by that team when performing the task, and is domain-dependent. In a dynamic world, the performance of teams of agents is non-deterministic, so multiple observations of the same team at the task may result in different values:

Definition 2.1.3. *The **performance** of a team $A \subseteq \mathcal{A}$ is $\mathcal{P}_A$ and is non-deterministic.*

Definition 2.1.4. *An **observation** o_A is a real value corresponding to an observed utility attained by the team $A \subseteq \mathcal{A}$, i.e., o_A is a sample of $\mathcal{P}_A$.*

For example, suppose that a disaster has occurred in an urban area (such as a city), and that multiple urban search-and-rescue (USAR) personnel have arrived on the scene to offer their aid. $\mathcal{A}$ is the set of all USAR personnel, and the task is saving lives and minimizing damage to the

city. Suppose $A_0 \subseteq \mathcal{A}$ is a USAR team that performs the task. The performance of the team is measured and forms the observation $o_{A_0} = 3.4$. However, due to the dynamic nature of the USAR task (for example, wind causing fires to spread), the observed performance of A_0 may be different if the task was repeated, i.e., o_{A_0} would be a different number each time A_0 performed the task.

Since the performance of a team is non-deterministic, we define the δ-optimal team:

Definition 2.1.5. *The **δ-optimal team** is the team $A^*_\delta \subseteq \mathcal{A}$ such that there exists some utility u where A^*_δ obtains a utility of at least u with probability δ, and the probability of any other team A doing so is at most δ:*

$$\mathbb{P}(\mathcal{P}_{A^*_\delta} \geq u) = \delta \text{ and } \mathbb{P}(\mathcal{P}_A \geq u) \leq \delta \; \forall A \subseteq \mathcal{A}$$

The goal is to find the δ-optimal team of agents $A^*_\delta \subseteq \mathcal{A}$, and we assume that δ is given as part of the domain information. The δ-optimality measure was designed in order to rank non-deterministic performance; when performance is deterministic (or only the mean is considered), comparing the performance of teams is done with the $\geq$ operator. In δ-optimality, δ determines whether a risky team or risk-averse team is preferred. For example, when $\delta = \frac{1}{2}$, only the mean performance is considered, and the δ-optimal team is equivalent to the optimal team with non-deterministic performance. When $\delta < \frac{1}{2}$, a high-risk, high-reward team is preferred, i.e., one that has a low probability of attaining a high performance. Conversely, when $\delta > \frac{1}{2}$, a low-risk, low-reward team is preferred, i.e., one that has a high probability of attaining a low performance.

2.2 Task-Based Relationships

In order to find the δ-optimal team A^*_δ, we want to create a model of how well agents work together at the task. In the social networks domain, social graphs are used for team formation, where an edge between a pair of agents indicates that the agents have a social relationship, and the weight of the edge indicates the communication cost between them [Lappas et al., 2009, Dorn and Dustdar, 2010]. We are interested in forming teams that perform well at a task, and hence we model the task-based relationships among the agents as a task-based graph, where agents are vertices in the graph and edges represent the task-based relationships.

One approach to the task-based graph is to use a fully-connected graph, where the weight of an edge indicates the **cost** of 2 agents working together. Figure 2.1a shows an example of a fully-connected task-based graph with 4 agents. The agent a_1 works better with a_2 than a_3, hence the lower edge weight of 1.2 between a_1 and a_2, compared to 4.9 between a_1 and a_3.

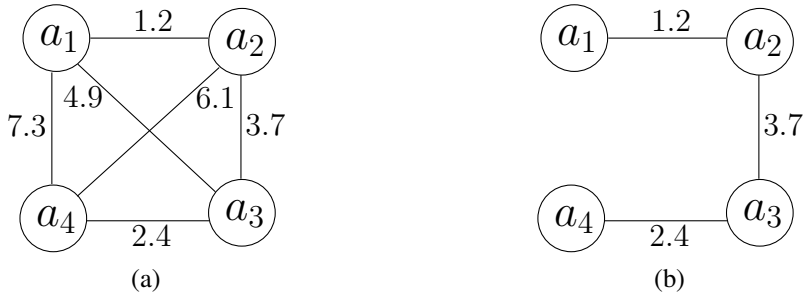


Figure 2.1: a) A fully-connected task-based graph with 4 agents, where the edge weights represent the cost of agents working together to perform the task. b) A connected task-based graph with the same 4 agents, where some edges have been removed while still preserving the pairwise distances between agents.

A fully-connected task-based graph has a major drawback — the task-based relationship between pairs of agents are completely independent, since every pair of agents is connected by an edge whose weight can be arbitrary, so the graph structure does not provide any additional information. Using the notion that edge weights represent the cost of agents working together, we introduce the concept of transitivity in task-based relationships. For example, if agent a_1 works very well with a_2, and a_2 works very well with a_3, then a_1 will work well with a_3. The transitivity occurs because for a_1 to work very well with a_2, there should be some underlying coordination strategy, and similarly between a_2 and a_3. Assuming that the agents use the same algorithms regardless of partners (i.e., they do not switch strategies), then a_1 and a_3 will be able to work well together since there is some overlap in their coordination strategies with a_2, albeit with higher cost. We assume that agents are always able to coordinate (e.g., all agents use the same communication protocol), or performance is based on their joint actions (similar to game theory), so any pair of agents has some task-based relationship.

To capture this notion of task-based transitivity, we use a connected graph where the shortest distance between agents indicates the cost of them working together. Figure 2.1b shows a connected task-based graph, by modifying the graph in Figure 2.1a such that edges $\{a_1, a_3\}$, $\{a_1, a_4\}$, and $\{a_2, a_4\}$ have been removed. However, the shortest distance between agents is equal in both Figure 2.1a and Figure 2.1b, and thus both graphs express the same task-based relationships. The edge weights in Figure 2.1a were chosen so that the pairwise distance is identical whether one edge (e.g., $\{a_2, a_4\}$) or two edges (e.g., $\{a_2, a_3\}$ and $\{a_3, a_4\}$) are traversed. We will consider other cases later.

While the shortest distance between agents in the task-based graph represents the cost of agents working together, we want to explicitly model the task-based relationship. Thus, we introduce a compatibility function $\phi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, where $\phi(d(a_i, a_j))$ returns the task-based

compatibility between agents a_i and a_j, and $d(a_i, a_j)$ is the shortest distance between them in the task-based graph. ϕ is a monotonically decreasing function, so larger distances correspond to lower compatibility. The compatibility function ϕ is domain-specific, and two intuitive examples of ϕ are:

$$\phi_{\text{fraction}}(d) = \frac{1}{d} \tag{2.1}$$

$$\phi_{\text{decay}}(d) = \exp\left(-\frac{d \ln 2}{h}\right) \tag{2.2}$$

where ϕ_{fraction} is a fraction function, and ϕ_{decay} is an exponential decay function with half-life h.

2.3 Agent Capabilities

The task-based graph and compatibility function described above model how well agents work together at a task. However, heterogeneous agents have different capabilities that affect their performance at a task. The performance of a team of agents depends on the capabilities of the agents *and* their task-based relationship.

One method to represent heterogeneous agent capabilities is to assign a value μ_i for each agent a_i, where μ_i corresponds to the agent a_i's mean capability at the task. In this thesis, we view capability as a measure of an agent's contribution to the team performance at a task, and not a binary (capable/incapable). As such, the mean capability refers to the average utility the agent contributes to the task, independent of its teammates. The effects of teammates on the team performance are modeled via the task-based graph.

Figure 2.2a shows an example of 3 agents a_1, a_2, a_3, where a_1 works equally well with agents a_2 and a_3, i.e., $d(a_1, a_2) = d(a_1, d_3)$. Even though a_1 works equally well with them, a_2 has a higher mean capability than a_3 — the mean performance of team $\{a_1, a_2\}$ is greater than $\{a_1, a_3\}$.

While the agents' mean capabilities at the task can be represented as values, they do not capture variability. Since the agents act in a dynamic, non-deterministic world, their performance varies over multiple instances. Thus, instead of a single value, we use a Normally-distributed random variable to represent the agent's capability, i.e., each agent a_i is associated with a variable C_i, which is the agent a_i's non-deterministic capability at the task. We use a Normal distribution because it is unimodal, corresponding to the agent's peak performance, and variability in the performance as its standard deviation. Also, Normal distributions are widely used for their mathematical properties, which we exploit later.

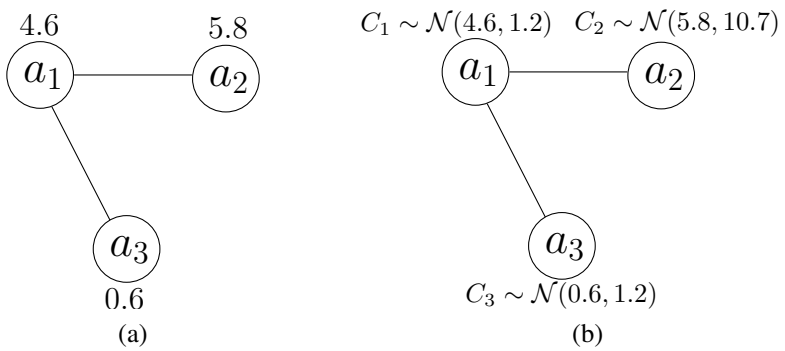


Figure 2.2: Task-based graphs with 3 agents, where the agents' heterogeneous capabilities are attached to each vertex and are represented as a) values; b) Normally-distributed variables.

By using a Normal variable, we can now model how consistent an agent is at the task. Figure 2.2b shows a modification of Figure 2.2a, where a_2 has a higher variance for its performance compared to a_3. As such, depending on δ, the team $\{a_1, a_3\}$ may outperform $\{a_1, a_2\}$.

2.4 Unweighted Synergy Graph Model

We have detailed how task-based relationships are represented with the compatibility function ϕ that uses the distance between agent vertices in a graph, and how agent capabilities are represented as Normally-distributed variables. We formally define the Unweighted Synergy Graph model and how it is used to compute the performance of a team of agents at a task.

Definition 2.4.1. *An **Unweighted Synergy Graph** is a tuple (G, C), where:*

- *$G = (V, E)$ is a connected unweighted graph,*
- *$V = \mathcal{A}$, i.e., the set of vertices corresponds to the set of agents,*
- *E are unweighted edges in G, and*
- *$C = \{C_1, \ldots, C_N\}$, where $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is agent a_i's capability.*

Synergy Graphs are connected, so at least one path exists between any pair of vertices. The distance $d(a_i, a_j)$ between any two agents a_i, a_j is defined to be the shortest distance between them in the graph. In the Unweighted Synergy Graph model, $d(a_i, a_j)$ corresponds to the minimum number of edges between the vertices (since edges are unweighted).

Definition 2.4.2. *The **pairwise synergy** between two agents a_i and a_j is:*

$$\mathbb{S}_2(a_i, a_j) \;=\; \phi(d(a_i, a_j)) \;\cdot\; (C_i \;+\; C_j)$$

where $d(a_i, a_j)$ is the shortest distance between a_i and a_j, and ϕ is the compatibility function.

We assume that C_i and C_j are independent for all i, j, and so the summation $C_i + C_j$ in the pairwise synergy function can be performed easily. This assumption is reasonable as the effect of agents working in a team is captured by the compatibility function and their distance in the Synergy Graph, so the variables representing their individual capabilities are independent. The pairwise synergy function between any two agents always exists, since we assume that the Synergy Graph is connected and that the task can be accomplished with any number of agents.

We use the graph structure to model the synergy among agents, specifically using the edges (that connect two agents) to compute the shortest distance between pairs of agents, and hence the pairwise synergy is the building block for the synergy of a team of agents. We relax this assumption of using the shortest distance in the graph in Chapter 6.

Using the pairwise synergy function, we now define the performance of a team of agents:

Definition 2.4.3. *The **synergy** of a set of agents $A \subseteq \mathcal{A}$ is the average of the pairwise synergy of its components:*

$$\mathbb{S}(A) \;=\; \frac{1}{\binom{|A|}{2}} \cdot \sum_{\{a_i, a_j\} \in A} \mathbb{S}_2(a_i, a_j)$$

Using the synergy definitions, the synergy of a team A is a Normally-distributed random variable $C_A \;\sim\; \mathcal{N}(\mu_A, \sigma_A^2)$. In particular:

$$\mu_A = \frac{1}{\binom{|A|}{2}} \sum_{a_i, a_j \in A} \phi(d(a_i, a_j)) \cdot (\mu_i + \mu_j) \tag{2.3}$$

$$\sigma_A^2 = \frac{1}{\binom{|A|}{2}^2} \sum_{a_i, a_j \in A} \phi(d(a_i, a_j))^2 \cdot (\sigma_i^2 + \sigma_j^2) \tag{2.4}$$

The synergy function $\mathbb{S}$ models the performance of teams of agents, and the synergy of a team $A \subseteq \mathcal{A}$ depends critically on the capabilities of the agents in A and the pairwise distances between them. The definition of synergy involves the summation of individual capabilities, and it is weighted by the distance of agents in the Synergy Graph. As such, the addition or removal of specific agents can have a large impact on the total score of a team. For example, from Figure 2.3, suppose that $\phi(d) = \frac{1}{d}$. Then, the team $\{a_1, a_3\}$ has a mean performance of 9.5, and the team $\{a_1, a_2, a_3\}$ (adding a_2 into $\{a_1, a_3\}$) increases the mean to 19.9. Conversely, the team $\{a_1, a_3, a_5\}$ has a mean of 14.4, even though the individual capability of a_5 is higher than a_2.

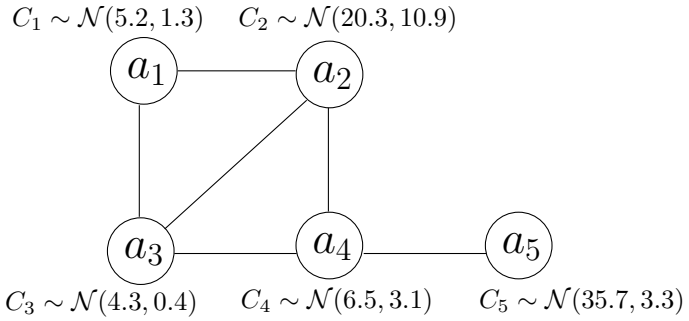


Figure 2.3: An Unweighted Synergy Graph with 5 agents. Each vertex represents an agent, and the distance between vertices in the graph indicate how well agents work together. Agent capabilities are modeled as Normally-distributed variables.

2.5 Weighted Synergy Graph Model

The Unweighted Synergy Graph model uses unweighted edges in its graph structure, and allows a conceptually simpler representation of task-based relationships. However, not all task-based relationships can be modeled with unweighted edges. Figure 2.4a shows one such example of a task-based graph that cannot be represented with unweighted edges. Suppose $\phi(d) = \frac{1}{d}$, the compatibility of $\{a_1, a_2\}, \{a_1, a_3\}$, and $\{a_2, a_3\}$ are then $\frac{1}{3}, \frac{1}{8}$ and $\frac{1}{5}$ respectively. Suppose we use an unweighted task-based graph and find an adjusted compatibility function ϕ', such that $\phi'(d'(a_i, a_j)) = \phi(d(a_i, a_j))$, where d' and d are the shortest distance functions in the unweighted and weighted task-based graphs respectively. In an unweighted task-based graph, the longest possible distance between 2 agents is $|\mathcal{A}| - 1$, if all the agents formed a chain. Thus, with 3 agents, the greatest distance is 2. Since the compatibility function is monotonically decreasing, $\phi'(2) = \frac{1}{8}$, which is the lowest compatibility among a_1, a_2, and a_3, and implies that $d'(a_1, a_3) = 2$. However, this in turn implies that $d'(a_1, a_2) = d'(a_2, a_3) = 1$ (Figure 2.4b). As such, no compatibility function ϕ' can be defined, since $\phi'(1)$ has to be equal to $\frac{1}{3}$ and $\frac{1}{5}$ which is impossible.

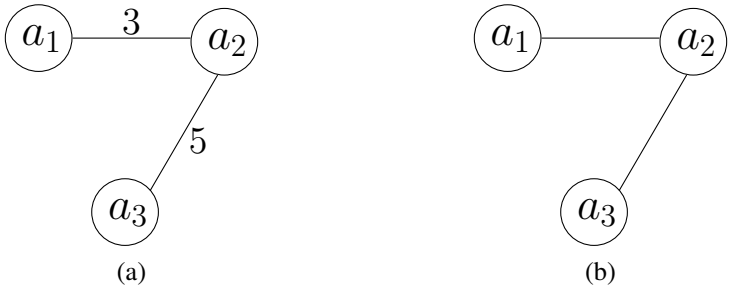


Figure 2.4: a) A weighted task-based graph with 3 agents. b) An unweighted task-based graph with the 3 agents (all edges have a weight of 1). If the compatibility function in (a) is $\phi(d) = \frac{1}{d}$, the task-based relationships of the agents cannot be represented with an unweighted graph and an adjusted compatibility function.

Thus, to model task-based relationships that require weighted edges, we define the Weighted Synergy Graph:

Definition 2.5.1. *The **Weighted Synergy Graph** is a tuple (G, C), where:*

- *$G = (V, E)$ is a connected weighted graph,*
- *$V = \mathcal{A}$, i.e., the set of vertices corresponds to the set of agents,*
- *$e_{i,j} = (a_i, a_j, w_{i,j}) \in E$ is an edge between agents a_i, a_j with weight $w_{i,j} \in \mathbb{R}^+$,*
- *$C = \{C_1, \ldots, C_N\}$, where $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is agent a_i's capability.*

The definitions of pairwise synergy (Definition 2.4.2) and synergy (Definition 2.4.3) apply to the Weighted Synergy Graph model, with the caveat that the shortest distance $d(a_i, a_j)$ between two agents a_i, a_j uses the weights on the edges.

2.5.1 A Weighted Synergy Graph Example

Figure 2.5 shows an example of a Weighted Synergy Graph with 5 agents in a rescue task. a_1, a_2 and a_3 are personnel in the ambulance, namely the agent that carries the stretcher, the agent that performs CPR, and the agent that drives the ambulance respectively. Since these 3 agents have trained as a team, their task-based relationship is high, and so the distance between them in the Weighted Synergy Graph is low. As heterogeneous agents, their individual capabilities are different. For example, the capability of a_2, the CPR agent, has a high mean (that reflects the high payoff CPR provides to the task) and correspondingly high variance (that reflects that CPR does not always succeed). As each agent is individually capable of saving lives (i.e., the driver agent and stretcher agent can also perform first aid), the task can be completed with *any* subset of agents.

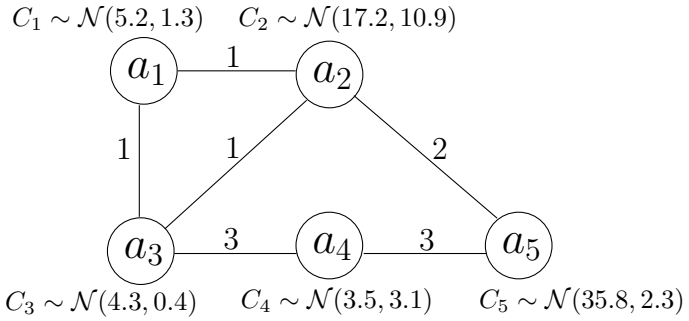


Figure 2.5: An example of a Weighted Synergy Graph modeling capabilities and the task-based relationships of a group of agents in a rescue task.

a_4 is an intern at the hospital, and a_5 is a surgeon at the hospital. The surgeon's capability is both better and more consistent than the CPR agent's: $C_5 \sim \mathcal{N}(35.7, 3.3)$ versus $C_2 \sim \mathcal{N}(20.3, 10.9)$, reflecting the surgeon's skills.

Using Definitions 2.4.2 and 2.4.3, and assuming that $\phi(d) = \frac{1}{d}$, we can compute the synergy of the agents. The team of agents in the ambulance, i.e., $\{a_1, a_2, a_3\}$, has a synergy of $C_{\{a_1,a_2,a_3\}} \sim \mathcal{N}(17.8, 2.8)$. Since the surgeon a_5 is more capable than a_2 (the CPR agent), one might consider replacing a_2 with a_5. However, the synergy would be $C_{\{a_1,a_3,a_5\}} \sim \mathcal{N}(12.1, 0.3)$, which has a lower mean than the team $\{a_1, a_2, a_3\}$. The lower mean is due to the task-based relationships among the agents — the other agents in the ambulance (the stretcher agent and the driver agent) work well and perform better with the CPR agent than with the surgeon.

When these four agents are in a team, their synergy is $C_{\{a_1,a_2,a_3,a_5\}} \sim \mathcal{N}(17.8, 0.8)$, which shows that the addition of the surgeon agent (instead of replacing the CPR agent) potentially benefits the team by lowering the variance of their performance. However, adding agents does not always improve the team performance — the team of all the agents has a synergy $C_{\{a_1,a_2,a_3,a_4,a_5\}} \sim \mathcal{N}(13.0, 0.3)$, since the intern agent does not contribute much to the performance and has a poor task-based relationship with the other agents, as reflected by its high edge weights to other agents.

Thus, the Weighted Synergy Graph captures interesting and complex relationships among agents, where the composition of the team makes a significant difference in the overall task performance.

2.5.2 Equivalence in Weighted Synergy Graphs

Figure 2.6 shows three examples of Weighted Synergy Graphs with three agents, such that the shortest distance between agents are: $d(a_1, a_2) = 1.3$, $d(a_1, a_3) = 2.4$, and $d(a_2, a_3) = 3.7$. Since the shortest distance is used to compute synergy, the three Weighted Synergy Graphs are

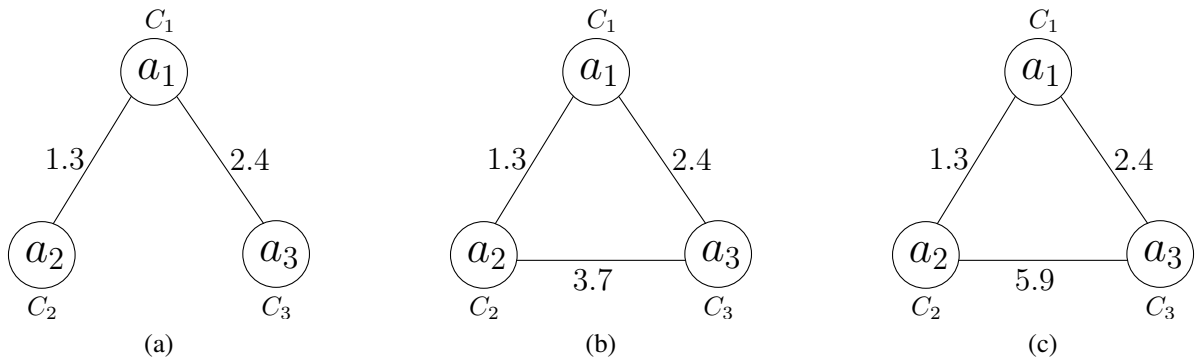


Figure 2.6: Three equivalent Weighted Synergy Graphs, i.e., the shortest distance between pairs of agents is equivalent in the three graphs.

equivalent. In Figure 2.6a, only 2 edges are present while 3 edges are present in Figures 2.6b and 2.6c. If the weight of edge $e_{2,3}$ in Figure 2.6b was less than 3.7, it would not be equivalent to the other two. Also, edge $e_{2,3}$ in Figure 2.6c is not used, since a shorter path exists between a_2 and a_3 using $e_{1,2}$ and $e_{1,3}$ — we consider how to use such edges in graphs for modeling non-transitive relationships later in Section 5.3. Thus, in general, more than one graph structure can be used to define the shortest distance relationship among agents.

Definition 2.5.2. *Weighted Synergy Graphs* $S = (G, C)$ *and* $S' = (G', C')$ *are* ***equivalent*** *if:*

- $V = V'$,
- $C = C'$,
- $d(a_i, a_j)$ *in graph* $G = d(a_i, a_j)$ *in graph* G' $\forall a_i, a_j \in \mathcal{A}$.

Since the shortest distance between all pairs of agents are identical, their compatibility (as computed by ϕ) is identical, and hence the synergy of a team of agents are equal given equivalent Weighted Synergy Graphs. In this work, we do not distinguish between equivalent Weighted Synergy Graphs.

2.6 Assumptions of the Synergy Graph Model

The Synergy Graph model captures the task-based relationships among agents and is used to compute the synergy of agent teams using the distances in the graph and the agent capabilities. We now list the assumptions of the model, and a short discussion of the rationales behind the assumptions and how the assumptions can be relaxed:

1. Team performance is Normally-distributed,
2. The synergy of a team is a function of the pairwise synergies,

3. Task-based relationships are transitive and modeled via shortest distance in the graph.

Regarding Assumption 1, Section 2.3 explains the rationale of using Normal distributions to model team performance. If the actual performance was from some other distribution, the Synergy Graph model can be modified to use that distribution or any arbitrary distribution, although the computation of pairwise synergy $\mathbb{S}_2$ and synergy $\mathbb{S}$ could be more complex (by adding arbitrary distributions together).

Assumption 2 comes about from the graph-based nature of the Synergy Graph model. Since edges involve two agents, we derive the synergy function $\mathbb{S}$ from the pairwise synergy $\mathbb{S}_2$. Further, the task can be accomplished by any number of agents, so pairwise synergy between any two agents always exists. If the assumption is relaxed, the Synergy Graph can be extended to use hyperedges (edges between more than two agents), and $\mathbb{S}$ can be updated to reflect the change.

The Synergy Graph model assumes transitivity in the task-based relationship (Assumption 3). Our work focuses on tasks where this assumption holds. In cases where the assumption does not hold, the model can be updated to use other measures other than the shortest distance between agents, which we consider later in Section 5.3.

2.7 Solving the Team Formation Problem

We defer how a Synergy Graph is learned from observations of performance to the next chapter. In this section, we explain how to use a Synergy Graph to form the δ-optimal team for the task, i.e., the team A^*_δ such that $\mathbb{P}(\mathcal{P}_{A^*_\delta} \geq u) = \delta$ and $\mathbb{P}(\mathcal{P}_A \geq u) \leq \delta \; \forall A \subseteq \mathcal{A}$.

Using the Synergy Graph model and the synergy equations, we can compute the synergy of any team of agents $A \subseteq \mathcal{A}$. However, the synergy computed is a Normally-distributed variable, and we need to rank such variables to choose one possible team over another.

To do so, we use an evaluation function that converts a Normally-distributed variable into a real number using the parameter δ [Liemhetcharat and Veloso, 2011]:

$$\texttt{Evaluate}(X, \delta) = \mu_X + \sigma_X \cdot \Phi^{-1}(1 - \delta) \tag{2.5}$$

where $X \sim \mathcal{N}(\mu_X, \sigma^2_X)$ and Φ^{-1} is the inverse of the cumulative distribution function of the standard Normal distribution.

In particular, for any Normally-distributed variable X, $\texttt{Evaluate}(X, \delta)$ returns a value v_X such that $P(X \geq v_X) = \delta$. When $\delta = \frac{1}{2}$, $\texttt{Evaluate}(X, \delta)$ returns μ_X, the mean of X; σ_X, the standard deviation of X, decreases (increases) the value returned by $\texttt{Evaluate}(X, \delta)$ when $\delta > \frac{1}{2}$ ($\delta < \frac{1}{2}$).

Theorem 2.7.1. *Let* $A, A' \subseteq \mathcal{A}$*, and* $C_A = \mathbb{S}(A), C_{A'} = \mathbb{S}(A')$.
Let $\texttt{Evaluate}(C_A, \delta) = v_A$ *and* $\texttt{Evaluate}(C_{A'}, \delta) = v_{A'}$.
If $v_A \geq v_{A'}$*, then* $P(C_{A'} \geq v_A) \leq \delta$.

Proof. $P(C_{A'} \geq v'_A) = \delta$ and $v_A \geq v'_A$.
$\Rightarrow P(C_{A'} \geq v_A) \leq P(C_{A'} \geq v_{A'})$
$\Rightarrow P(C_{A'} \geq v_A) \leq \delta$ □

Collorary 2.7.2. $A^*_\delta = \text{argmax}_{A \subseteq \mathcal{A}} \texttt{Evaluate}(\mathbb{S}(A), \delta)$

Proof. Let $A_\delta = \text{argmax}_{A \subseteq \mathcal{A}} \texttt{Evaluate}(\mathbb{S}(A), \delta)$.
Let $\texttt{Evaluate}(\mathbb{S}(A_\delta), \delta) = v$.
$\forall A \subseteq \mathcal{A}$, $P(\mathbb{S}(A) \geq v) \leq \delta$ (from Theorem 2.7.1)
$\therefore A^*_\delta = A_\delta$ □

Hence, `Evaluate` returns a real number that we use to rank possible teams and find the δ-optimal team A^*_δ, since the team that returns the highest value from `Evaluate` corresponds to the δ-optimal team.

We now contribute two team formation algorithms: `Find`δ`OptimalTeam`, a branch-and-bound algorithm that finds the δ-optimal team in exponential time, and a second algorithm `Approx`δ`OptimalTeam`, that approximates that δ-optimal team in polynomial time.

Both algorithms assume that $n^* = |A^*_\delta|$ is known and given as a parameter to the algorithm. This is a reasonable assumption, since the size of teams are typically limited by external factors, e.g., a cost budget, size restrictions. For example, the size of teams in sports is fixed, and also in tasks that require handing of a fixed number of devices, such as the operators of an ambulance. If n^* is unknown, then the algorithms are run iteratively for $n = 1, \ldots, N$ and the best-performing team is selected. Both algorithms are applicable to the Unweighted Synergy Graph and Weighted Synergy Graph models without any modifications, since the synergy function $\mathbb{S}$ applies to both models.

2.7.1 Forming the δ-Optimal Team

Our first team formation algorithm, `Form`δ`OptimalTeam`, uses branch-and-bound to find the δ-optimal team. Algorithm 1 shows the pseudocode of the algorithm. The inputs to the algorithm are n (the size of the desired team), δ (for δ-optimality), S (the Synergy Graph), A (the team being considered), and v_{best} (the value of the best team found so far). $|A| \leq n$ during the execution of the algorithm, and contains the fixed members of the team, e.g., if $n = 5$ and $A = \{a_1, a_2\}$, then

`Form`δ`OptimalTeam` returns the optimal team of 5 agents given that $\{a_1, a_2\}$ are in the team. The initial call to `Form`δ`OptimalTeam` sets $A = \emptyset$ and $v_{\text{best}} = -\infty$.

Algorithm 1 Find the δ-optimal team of size n

FormδOptimalTeam($n, \delta, S, A, v_{\text{best}}$)

```
 1: if |A| = n then
 2:    v_A ← Evaluate(𝕊(A), δ)
 3:    if v_A ≥ v_best then
 4:       return (A, v_A)
 5:    else
 6:       return (A_best, v_best)
 7:    end if
 8: end if
 9: j ← max_{a_i ∈ A} (i)
10: for i = j + 1 to N do
11:    A_next ← A ∪ {a_i}
12:    (next_min, next_max) ← CalculateBounds(n, δ, S, A_next)
13:    if next_max ≥ v_best then
14:       v_best ← max(v_best, next_min)
15:       (A_best, v_best) ← Formδ OptimalTeam(n, δ, S, A_next, v_best)
16:    end if
17: end for
18: return (A_best, v_best)
```

Lines 1–8 describe the base case when the team is fully-formed — the value of the team is computed with `Evaluate`, and the team is returned if its value is greater than the current best team. Otherwise, branching and bounding is performed. The branching occurs as agents are added to A and the algorithm is recursively called (lines 11 and 15). The bounds of the team are computed with `CalculateBounds` (described below), and the team is pruned if the maximum of the bound (next_{max}) is lower than the current best value v_{best} (line 13).

To compute the bounds of a team given a Synergy Graph, `CalculateBounds` uses the following heuristic. The minimum and maximum pairwise distance between vertices in the Synergy Graph (excluding pairs of the selected team A_{next}) are computed, as well as the minimum and maximum agent capabilities (excluding the agents in A_{next}). The maximum bound is computed by using the synergy function $\mathbb{S}$ and assuming that all distances with undetermined agents are the minimum distance, and agent capabilities are maximum. Similarly, the minimum bound is computed using the maximum distances and minimum agent capabilities.

Finding the δ-optimal team is NP-hard, and branch-and-bound can fully explore the space in the worst case. As such, the runtime of `Form`δ`OptimalTeam` is $O(N^n)$, where N is the total number of agents, and n is the number of agents in the team. If n^* is unknown, then the algorithm is run for increasing n for a total runtime of $O(N^N)$.

2.7.2 Approximating the δ-Optimal Team

Finding the δ-optimal team takes exponential time in the worst case, and in many situations, a near-optimal team is sufficient to solve the problem. Algorithm 2 shows the pseudocode for `Approx`δ`OptimalTeam`, that approximates the δ-optimal team of size n, given δ and a Synergy Graph S.

Algorithm 2 Approximate the δ-optimal team of size n

ApproxδOptimalTeam(n, δ, S)

1: $A_{\text{best}} \leftarrow$ `RandomTeam`(S, n)
2: $v_{\text{best}} \leftarrow$ `Evaluate`$(\mathbb{S}(A_{\text{best}}), \delta)$
3: **repeat**
4: $\quad A_{\text{neighbor}} \leftarrow$ `NeighborTeam`(A_{best})
5: $\quad v_{\text{neighbor}} \leftarrow$ `Evaluate`$(\mathbb{S}(A_{\text{neighbor}}), \delta)$
6: $\quad$ **if** `accept`$(v_{\text{best}}, v_{\text{neighbor}})$ **then**
7: $\quad\quad A_{\text{best}} \leftarrow A_{\text{neighbor}}$
8: $\quad\quad v_{\text{best}} \leftarrow v_{\text{neighbor}}$
9: $\quad$ **end if**
10: **until** `done`()
11: **return** A_{best}

Algorithm 2 first begins by generating a random team of size n, which is performed by randomly selecting n agents from $\mathcal{A}$. The value of the team is then computed with the `Evaluate` function. The random team and its value thus forms the initial guess of the algorithm.

Next, the algorithm begins its approximation loop. Lines 3–10 of Algorithm 2 show a general approximation algorithm (with functions `accept` and `done`) to illustrate that our algorithm is compatible with most approximation algorithms. In this thesis, we use simulated annealing but other approximation algorithms (such as hill-climbing) are suitable as well. In simulated annealing, the `done` function would check if the desired number of iterations has been run, and the `accept` function would accept a neighbor based on the temperature schedule (computed from the current iteration number) and the difference in `Evaluate` scores of the current best guess and its neighbor.

`NeighborTeam`(A_{best}) randomly swaps one selected agent $a \in A_{\text{best}}$ with an unselected one $a' \in \mathcal{A} \setminus A_{\text{best}}$. In this way, neighbor teams are generated from the current best estimate A_{best}

so as to effectively explore the space of possible teams of size n. The value of the neighbor team v_{neighbor} is computed with `Evaluate`, and the team is accepted or rejected based on the criteria of the approximation algorithm (e.g., simulated annealing uses a temperature schedule).

Thus, Algorithm 2 finds an approximation to the δ-optimal team given its size n. The algorithm runs in $O(n^2)$ (the synergy function $\mathbb{S}$ takes $O(n^2)$ and simulated annealing runs a constant number of iterations) if n^* is known. Otherwise, the algorithm is run iteratively for increasing n and has total runtime of $O(N^3)$. In comparison, a brute-force algorithm would take $O(\binom{N}{n})$ if n^* is known, and $O(2^N)$ otherwise.

2.7.3 Comparing the Team Formation Algorithms

To evaluate both team formation algorithms, and compare their performance (amount of the search space explored, and value of the formed team), we generated random Weighted Synergy Graphs. We varied the number of agents in the graph from 10 to 15, and randomly created 1000 connected weighted graph structures where each edge weight was an integer that varied from 1 to 5. For each weighted graph structure generated, the agent capabilities $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ were also randomly generated, such that $\mu_i \in (50, 150)$ and $\sigma_i^2 \in (0, 10000)$. The size of the desired team n was set to $\lfloor \frac{N}{2} \rfloor$, so that the search space is as large as possible.

`Form`δ`OptimalTeam` always finds the optimal team, so we were interested in evaluating how many times `CalculateBounds` and `Evaluate` were called. For the other algorithm `Approx`δ`OptimalTeam`, we ran simulated annealing for 1000 iterations (so 1000 calls to `Evaluate` were made), and we were interested in evaluating the quality of the team formed.

Table 2.1 shows the results of our comparisons. The effectiveness of the formed team is expressed as a value in $[0, 1]$, where 0 means the worst possible team (with the minimum value), and 1 means the optimal team (with the maximum value):

$$\text{Effectiveness}(A) = \frac{\texttt{Evaluate}(A, \delta) - \texttt{Evaluate}(A_\delta^{\min}, \delta)}{\texttt{Evaluate}(A_\delta^*, \delta) - \texttt{Evaluate}(A_\delta^{\min}, \delta)} \tag{2.6}$$

where A_δ^* is the δ-optimal team, and $A_\delta^{\min}$ is the worst possible team given δ.

`Form`δ`OptimalTeam` finds the optimal team but evaluates a large number of teams — the number of calls to `CalculateBounds` and `Evaluate` are greater than the size of the search space. In comparison, running `Approx`δ`OptimalTeam` for a fixed number of iterations (1000) finds the δ-optimal team or a team very close to optimal most of the time. When there are twelve or more agents, `Approx`δ`OptimalTeam` performs competitively with `Form`δ`OptimalTeam` while evaluating a smaller amount of teams (only 1000). We believe the high performance of

Weighted Synergy Graph						
# agents	FormδOptimalTeam			ApproxδOptimalTeam		
	# Evaluations	Time (ms)	Effectiveness	# Evaluations	Time (ms)	Effectiveness
10	340 ± 69	18 ± 4	1	1000	17 ± 2	0.996 ± 0.021
11	545 ± 125	32 ± 8	1	1000	17 ± 2	0.992 ± 0.034
12	1216 ± 254	83 ± 18	1	1000	24 ± 2	0.997 ± 0.020
13	1993 ± 449	157 ± 37	1	1000	30 ± 2	0.996 ± 0.021
14	4439 ± 932	412 ± 90	1	1000	62 ± 2	0.998 ± 0.010
15	7307 ± 1694	791 ± 191	1	1000	101 ± 3	0.998 ± 0.013

Table 2.1: The number of evaluations done by the algorithms FormδOptimalTeam and ApproxδOptimalTeam to compute and approximate the δ-optimal team respectively in a Weighted Synergy Graph, the time taken by the algorithms in milliseconds, and the quality of the team found (where 0 means the worst team and 1 is the optimal team).

Unweighted Synergy Graph						
# agents	FormδOptimalTeam			ApproxδOptimalTeam		
	# Evaluations	Time (ms)	Effectiveness	# Evaluations	Time (ms)	Effectiveness
10	265 ± 64	14 ± 4	1	1000	17 ± 3	0.998 ± 0.013
11	391 ± 108	23 ± 6	1	1000	17 ± 2	0.997 ± 0.016
12	860 ± 233	56 ± 16	1	1000	24 ± 2	0.999 ± 0.008
13	1291 ± 389	99 ± 31	1	1000	30 ± 2	0.998 ± 0.011
14	2849 ± 811	262 ± 77	1	1000	62 ± 2	0.999 ± 0.004
15	4237 ± 1346	451 ± 149	1	1000	101 ± 3	0.999 ± 0.005

Table 2.2: The number of evaluations done by the algorithms FormδOptimalTeam and ApproxδOptimalTeam in an Unweighted Synergy Graph, the time taken in milliseconds, and the effectiveness of the team found.

FormδOptimalTeam is because the neighbor generation allows for good exploration of the space — a team with a high score will remain with a high score when a single member is swapped. Comparatively, in FormδOptimalTeam, the bounds of performance are computed when some agents in the team is fixed; the bounds are large when few agents are fixed and only become narrow as most agents are fixed. As such, pruning can only occur towards the bottom of the branch-and-bound search tree and so a large search space is required. Thus, we use ApproxδOptimalTeam for the remainder of this thesis. Similarly, FormδOptimalTeam takes much larger amounts of time to run as the number of agents increases, compared to ApproxδOptimalTeam. In particular, although ApproxδOptimalTeam performs 1000 evaluations, there is an overhead to computing the shortest distance between agents that explains why there is an increase in the computation time as the number of agents increases.

We repeated the experiment using Unweighted Synergy Graphs, and the results were similar. Table 2.2 shows the results of the second set of experiments.

2.8 Comparing Unweighted and Weighted Synergy Graphs

In this section, we evaluate the expressiveness of the Weighted Synergy Graph model compared to the Unweighted Synergy Graph. To do so, we assume that the agent capabilities are known, and we iterate across possible Synergy Graph structures.

2.8.1 Experimental Setup

The `LearnSynergyGraph` algorithm (explained in Chapter 3) learns the structure of the Weighted Synergy Graph, and we use this algorithm in these experiments, except that the function `LearnCapabilities` is not called since the agent capabilities are known. The learning algorithm iteratively modifies the Synergy Graph structure, and computes the log-likelihood of the observation set given the Synergy Graph.

We varied the number of agents N from 5 to 15. In each trial, a *hidden* Weighted Synergy Graph was randomly created where the agent capabilities $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ were generated with $\gamma = 10$, where $\mu_i \in (\frac{\gamma}{2}, \frac{3\gamma}{2})$ and $\sigma_i^2 \in (0, \gamma^2)$. The hidden Weighted Synergy Graph is hidden from the learning and team formation algorithms, and is used as the ground truth of the experiment. The observation set comprising the performance of teams with 2 and 3 agents are extracted from the hidden Weighted Synergy Graph. From the same observation set, a Weighted Synergy Graph and an Unweighted Synergy Graph are learned. The performance of teams with 4 and more agents are extracted from the hidden Weighted Synergy Graph and form the test observation set. We calculate the log-likelihood of the learned Weighted Synergy Graph and Unweighted Synergy Graph using the test set.

In addition, with each of the learned Synergy Graphs, we approximate the δ-optimal team of size $n^* = \max(4, \lfloor \frac{N}{2} \rfloor)$ using `Approx`δ`OptimalTeam`. We chose such a value of n^* so as to maximize the number of possible teams (i.e., $\binom{N}{\lfloor \frac{N}{2} \rfloor} \geq \binom{N}{i}\ \forall i \in \{1, \ldots, N\}$), while having a minimum size of 4 since the performance of teams with sizes 2 and 3 are used in learning the Synergy Graphs. From the teams found, we computed their effectiveness (Equation 2.6).

Thus, the effectiveness of a team is a value from 0 to 1, where 1 is the δ-optimal team, and 0 is the worst-possible team. We use the effectiveness to measure the performance of the learned Synergy Graph, because the goal of the Synergy Graph is to form an effective team, and the performance is scaled since the actual performance of teams varies from trial to trial due to the randomized generation of the hidden Weighted Synergy Graph.

2.8.2 Comparison Results

In our first set of experiments, we used the fraction compatibility function ϕ_{fraction} (Equation 2.1), and set $\delta = \frac{1}{2}$. For each value of N, we performed 100 trials, where a different hidden Weighted Synergy Graph was generated in each trial. The Weighted and Unweighted Synergy Graphs were then learned using 1000 iterations of simulated annealing for each trial using the observation set extracted from the hidden model. Figure 2.7 shows the effectiveness of the teams found by the Weighted and Unweighted Synergy Graphs. As the number of agents N increases, the effectiveness of the teams found by both Synergy Graph types decrease, reflecting the increase in difficulty in learning the graph and finding the δ-optimal team. However, across all values of N, the team found by the learned Weighted Synergy Graph outperforms the team found by the learned Unweighted Synergy Graph. We performed a paired Student's T-test (one-tailed) on the 1100 total trials (11 values of N with 100 trials per N), and the results were statistically significant to a value of $p = 9.8 \times 10^{-258}$.

We repeated the experiments with $\phi_{\text{fraction}}(d)$, but increased the number of iterations of simulated annealing to 2000, to investigate if a higher number of iterations would improve the overall performance of the learned Synergy Graphs. Figure 2.8 shows the average effectiveness of the teams found by the Weighted and Unweighted Synergy Graphs with both 1000 and 2000 iterations of simulated annealing. The learned Weighted Synergy Graph performs better with 2000 iterations compared to 1000 iterations (statistically significant to a value of $p = 8.7 \times 10^{-20}$). However, a greater number of iterations of simulated annealing does not affect the performance of the learned Unweighted Synergy Graph ($p = 0.14$). Thus, a greater number of iterations of simulated annealing allows the Weighted Synergy Graph learning algorithm to converge on a closer match to the hidden Synergy Graph, while the "best" Unweighted Synergy Graph is already found within 1000 iterations and hence increasing the number of iterations has little effect.

Thirdly, we used the decay compatibility function $\phi_{\text{decay}}(d)$ (Equation 2.2) with half-life $h = 2$. In this way, the compatibility function ϕ_{decay} decreases at a slower pace than ϕ_{fraction} initially, but has much smaller values once d is large. As before, we varied N and ran 100 trials per value of N. We ran 1000 iterations of simulated annealing for both learning algorithms, and Figure 2.9 shows the effectiveness of the teams found by the learned Synergy Graphs. While the effectiveness of the learned Weighted Synergy Graph decreases more rapidly as N increases compared to ϕ_{fraction}, the learned Weighted Synergy Graph outperforms the learned Unweighted synergy Graph, with $p = 3.6 \times 10^{-160}$.

Thus, these experiments show that the Weighted Synergy Graph is more expressive than the Unweighted Synergy Graph. The results are statistically significant across agent sizes, with two compatibility functions, and with different maximum iterations of simulated annealing.

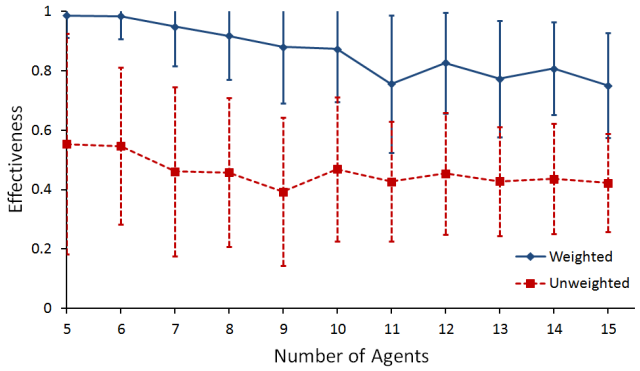


Figure 2.7: Effectiveness of teams found in the learned Weighted Synergy Graph and learned Unweighted Synergy Graph, using simulated annealing with 1000 iterations. The compatibility function was $\phi_{\text{fraction}}(d) = \frac{1}{d}$.

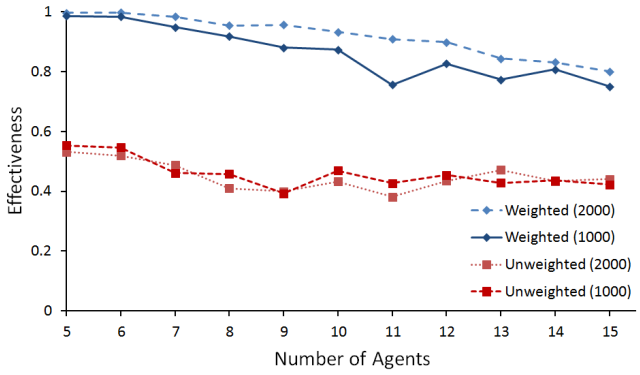


Figure 2.8: Average effectiveness of teams found in the learned Weighted and Unweighted Synergy Graphs, using 1000 and 2000 iterations of simulated annealing to learn the Synergy Graph structure, using $\phi_{\text{fraction}}(d) = \frac{1}{d}$.

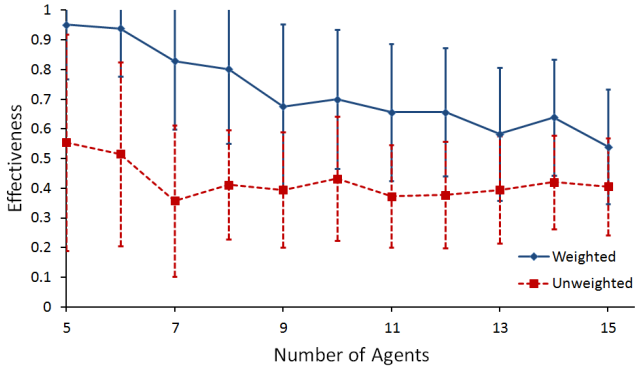


Figure 2.9: Effectiveness of teams found in the learned Weighted and Unweighted Synergy Graphs with $\phi_{\text{decay}}(d) = \exp\left(-\frac{d \ln 2}{2}\right)$, and 1000 iterations of simulated annealing.

2.9 Chapter Summary

This chapter presented the team formation problem, and formally defined the Unweighted Synergy Graph and Weighted Synergy Graph models, how the synergy of teams are computed using the models. This chapter presented `Form`δ`OptimalTeam` that forms the δ-optimal team in exponential time, and `Approx`δ`OptimalTeam` that approximates the δ-optimal team in polynomial time. Both algorithms are applicable to the Unweighted and Weighted Synergy Graph models, and `Approx`δ`OptimalTeam` finds near-optimal team without exploring a large amount of the search space. This chapter presented results that compared Unweighted and Weighted Synergy Graphs, and showed that teams formed from learned Weighted Synergy Graphs are more effective (i.e., closer to optimal) than teams formed from learned Unweighted Synergy Graphs.

Chapter 3

Learning Synergy Graphs

The previous chapter presented the Synergy Graph model, defined the synergy of multi-agent teams, and presented team formation algorithms that form effective teams. The team formation algorithms assume the existence of a Synergy Graph, but it is difficult, even for a domain expert, to manually write down a Synergy Graph based on a problem setup. This chapter presents `LearnSynergyGraph`, a learning algorithm that learns Synergy Graphs from observational data [Liemhetcharat and Veloso, 2012a, Liemhetcharat and Veloso, 2013d], so that the learned Synergy Graph can be used by the team formation algorithms. By automating the learning process, all that is required to apply the Synergy Graph model to real world scenarios is the collection of the observations. `LearnSynergyGraph` has two main components, a graph structure learner and an agent capability learner. This chapter presents two capability learners, `LearnCapabilityLeastSquares` and `LearnCapabilityNonLinear`, that `LearnSynergyGraph` uses to learn the agent capabilities using a least-squares solver and non-linear solver respectively.

3.1 Overview of the Learning Algorithm

The performance of multi-robot teams is initially unknown, and the Synergy Graph aims to model the performance with the synergy function $\mathbb{S}$. In order to do so, we assume that observations of the performance of some teams are available, that is representative of the overall team performance.

Recall that an observation o_A is a real value corresponding to an observed utility attained by agents $A \subseteq \mathcal{A}$, i.e., o_A is a sample of A's performance.

Definition 3.1.1. *An observation group $O_A = \bigcup o_A$ is the set of all observations of A.*

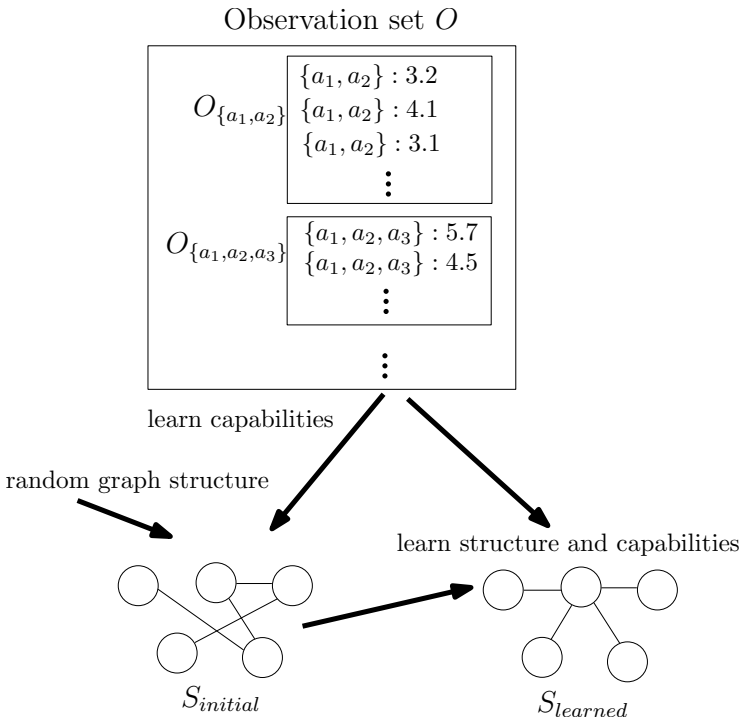


Figure 3.1: The process of learning from observations. The individual capabilities of agents in the Synergy Graphs are not shown.

Each observation group O_A contains all the observations of a unique team A. We assume that $\mathcal{A}$, the set of all agents, is known *a priori*. Otherwise, $\mathcal{A} = \bigcup_{O_A} A$. From these observation groups O_A, we define the observation set:

Definition 3.1.2. *The **observation set** O is the union of observation groups, i.e., $O = \bigcup \{O_A\}$.*

The observation set O is the only input to the learning algorithms — the algorithms do not require any other information, other than domain knowledge such as $\mathcal{A}$. Figure 3.1 shows an overview of the entire learning process.

Algorithm 3 shows the pseudocode of `LearnSynergyGraph`, the Synergy Graph learning algorithm that learns a Synergy Graph using only the observation set. Algorithm 3 first creates a random Synergy Graph structure, and learns the agent capabilities C using the Synergy Graph structure G and observation set O, which forms the initial guess of the Synergy Graph $S = (G, C)$. The log-likelihood of the observations given S is then computed.

Next, the algorithm enters the learning loop, where the learned Synergy Graph is iteratively improved. From the current guess of the Synergy Graph, a neighbor structure (i.e., the graph structure without agent capabilities) is generated. The neighbor generation function depends on the type of Synergy Graph being generated, which we elaborate later. From the neighbor structure G' and observation set O, agent capabilities C' are computed, which form a new Synergy

Graph S'. The log-likelihood of O given S' is computed, and S' is accepted as the best guess based on the approximation algorithm, as the goal is to find the Synergy Graph that best matches the observations, i.e., that Synergy Graph that is most likely to have produced the observations given its graph structure and individual capabilities. In this thesis, we use simulated annealing, so the `accept` function uses the difference in log-likelihoods of S and S', and the temperature schedule to decide if S' is accepted, and `done` considers if the maximum number of simulated annealing iterations have been performed.

Algorithm 3 Learn a Synergy Graph from observations

LearnSynergyGraph(O)

1: $G = (V, E) \leftarrow$ `RandomStructure`($\mathcal{A}$)
2: $C \leftarrow$ `LearnCapabilities`(G, O)
3: $S \leftarrow (G, C)$
4: score $\leftarrow$ `LogLikelihood`(S, O)
5: **repeat**
6: $G' = (V, E') \leftarrow$ `NeighborStructure`(G)
7: $C' \leftarrow$ `LearnCapabilities`(G', O)
8: $S' \leftarrow (G', C')$
9: score$'$ $\leftarrow$ `LogLikelihood`(S', O)
10: **if** `accept`(score, score$'$) **then**
11: $S \leftarrow S'$
12: score $\leftarrow$ score$'$
13: **end if**
14: **until** `done`()
15: **return** S

Hence, the learning algorithms consist of two main components — learning the Synergy Graph structure, and learning agent capabilities using the structure and observation set. We now elaborate on each of these components.

3.2 Learning the Synergy Graph Structure

The Synergy Graph learning algorithm, `LearnSynergyGraph`, first begins with a random Synergy Graph structure (with `RandomStructure`), and iteratively improves the structure by making random neighbors to the existing structure (with `NeighborStructure`) and comparing its log-likelihood. Every time a Synergy Graph structure is created, the agent capabilities are learned to completely define the Synergy Graph, so that the log-likelihood of the observations can be computed.

3.2.1 Generating a Random Synergy Graph Structure

The Synergy Graph model consists of a graph structure and agent capabilities. The Synergy Graph structure refers to the graph component of the model. Algorithm 4 generates a random Synergy Graph Structure. A common characteristic among the Synergy Graph models is that the graph is connected, i.e., any pair of vertices are connected, so line 3 loops until a connected set of edges are created. A domain variable EdgeProbability $\in (0, 1]$ determines the probability that a random edge is created, e.g., EdgeProbability $= 1$ creates a fully-connected graph.

Algorithm 4 Create a Random Synergy Graph Structure

RandomStructure($\mathcal{A}$)

```
1:  V ← A // Create vertices for all agents
2:  E ← ∅
3:  while not Connected(V, E) do
4:     E ← ∅
5:     for all v1, v2 ∈ V do
6:        if random() > EdgeProbability then
7:           if isUnweightedSynergyGraph() then
8:              E ← E ∪ {v1, v2}
9:           else if isWeightedSynergyGraph() then
10:             w ← randint(w_min, w_max)
11:             E ← E ∪ {v1, v2, w}
12:          end if
13:       end if
14:    end for
15: end while
16: return (V, E)
```

The functions `isUnweightedSynergyGraph()` and `isWeightedSynergyGraph()` are mutually exclusive, so only one function returns true, depending of which Synergy Graph model is being created. Weights are generated for the edges of Weighted Synergy Graphs, where `randint`$(w_{\text{min}}, w_{\text{max}})$ returns an integer between the two parameters w_{min} and w_{max} (inclusive).

3.2.2 Generating a Neighbor Synergy Graph Structure

`NeighborStructure` takes an existing Synergy Graph structure and makes a discrete change in order to generate a neighbor Synergy Graph Structure, by performing one of the following four possible actions:

1. Add an edge between two vertices
2. Remove a random edge that does not disconnect the graph

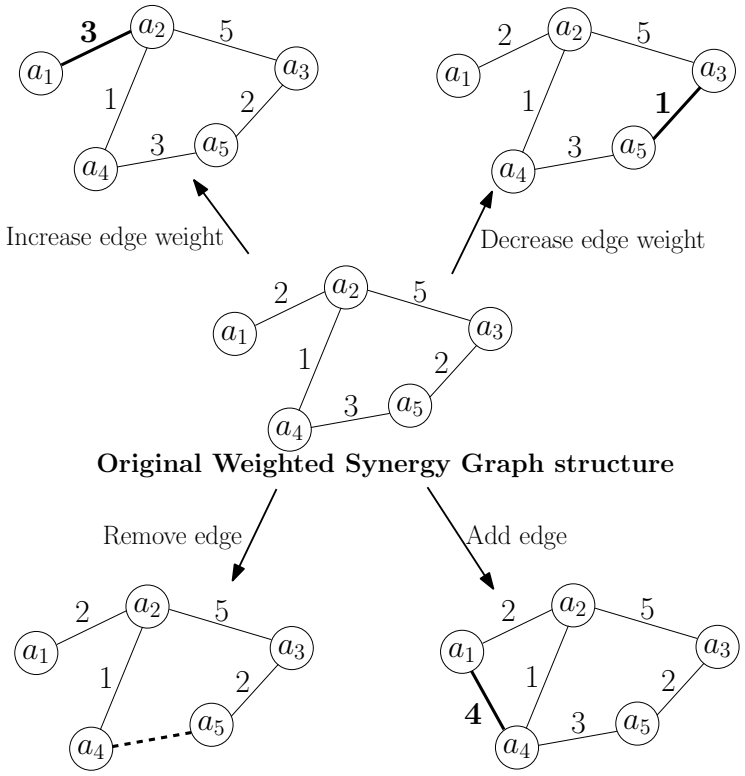


Figure 3.2: The four possible actions used to generate neighbor Weighted Synergy Graph structures.

3. Increase the weight of a random edge by 1
4. Decrease the weight of a random edge by 1

Only actions 1 and 2 are applicable for the Unweighted Synergy Graph model; all four actions are applicable to the Weighted Synergy Graph model (action 1 generates a random edge weight). Since the Synergy Graph structure is always a connected graph, action 2 will only be applied if removing the edge does not disconnect the graph. Actions 3 and 4 change the edge weights of the edges. While the edge weights in the Weighted Synergy Graph definition are real numbers, `NeighborStructure` only generates structures with integer weights. We use integer weights in the learning algorithm, as they provide a close approximation while still allowing discrete steps in neighbor generation. Higher accuracy can be achieved by increasing/decreasing the edge weights with smaller step size values. However, decreasing the step size increases the space of possible weighted graphs, so it is a trade-off that has to be managed. Figure 3.2 illustrates these four actions on an example Weighted Synergy Graph.

By iteratively changing the Synergy Graph structure, the Synergy Graph learning algorithm explores the space of possible structures in order to converge on the best structure that fits the observation set.

3.3 Learning Capabilities

After a Synergy Graph structure is generated, the agent capabilities are learned using the generated Synergy Graph structure and observation set. We present two agent learning algorithms, `LearnCapabilitiesLeastSquares`, that learns the agent capabilities with a least-squares solver, and `LearnCapabilitiesNonLinear`, that learns the agents capabilities with a non-linear solver. The latter algorithm uses less training data ($\mathcal{O}(N)$ compared to $\mathcal{O}(N^3)$ for the former), but requires a higher runtime due to the non-linear solver.

3.3.1 Learning Capabilities with a Least-Squares Solver

`LearnCapabilitiesLeastSquares` learns the individual agent capabilities, using the Synergy Graph structure G and observation set O, and Algorithm 5 shows the pseudocode. Matrices $\mathfrak{M}$ and b are created such that $\mathfrak{M}x = b$, e.g., $\mathfrak{M}_\mu\, x_\mu = b_\mu$, where $x_\mu = [\mu_{a_1}, \ldots, \mu_{a_N}]^T$. Using Equations 2.3 and 2.4, each row in $\mathfrak{M}$ and b corresponds to an observation group O_A in O. The only unknowns in the equations are the agent capabilities (the compatibility function ϕ are known and pairwise distances are computed from the Synergy Graph structure), so the equations are manipulated to make the agent capabilities the subject. Each column in $\mathfrak{M}$ corresponds to an agent in $\mathcal{A}$. A least-squares solver is run to find x, which corresponds to the means and variances of the agent's capabilities.

We assume that O, the observation set, contains all pairs and triples of agents. Let $\mathcal{A}_2 \subset 2^{\mathcal{A}}$ such that $\mathcal{A}_2 = \bigcup_{A \in \mathcal{A} \textbf{ s.t. } |A|=2} A$. Similarly, let $\mathcal{A}_3 \subset 2^{\mathcal{A}}$ such that $\mathcal{A}_3 = \bigcup_{A \in \mathcal{A} \textbf{ s.t. } |A|=3} A$. Hence, $\mathcal{A}_2$ and $\mathcal{A}_3$ are the sets of all pairs and triples of agents respectively. Our learning algorithm uses the observations of the agents in $\mathcal{A}_{2,3} = \mathcal{A}_2 \cup \mathcal{A}_3$. Specifically, let O be the set of observations, where $\forall A \in \mathcal{A}_{2,3}, \exists O_A \in O$ such that each o_A is an observation of the performance of the team A (we typically use $|O_A| = 30$ in this thesis, although any large $|O_A|$ would suffice). Since any subset of agents will attain a performance value at the task, and the synergy function $\mathbb{S}$ is computed from the pairwise synergy function $\mathbb{S}_2$, the observation set O is sufficient for learning. In particular, $\mathcal{A}_2$ provides information about the shortest distance between pairs of agents and the agents' capabilities using the pairwise synergy function $\mathbb{S}_2$ (Definition 2.4.2). However, there are multiple solutions for any pairwise synergy (increasing capabilities versus decreasing distances), and $\mathcal{A}_3$ provides information about the overall structure of the graph using the synergy function $\mathbb{S}$ (Definition 2.4.3), and provides additional constraints to the learning problem.

For example, Figure 3.3 shows an example Weighted Synergy Graph structure, and the process of Algorithm 5. We will use the example Weighted Synergy Graph and the team $\{a_1, a_2\}$ to explain how the matrices are set. Suppose the compatibility function is ϕ_{fraction} (Equation 2.1).

Algorithm 5 Learn the agent capabilities with a least-squares solver

LearnCapabilitiesLeastSquares(G, O)

Let $O = \left\{O_{A_1}, \ldots, O_{A_{|O|}}\right\}$, where $A_i \subseteq \mathcal{A}$.
// Initialize the matrices with 0s
$\mathfrak{M}_\mu \leftarrow \mathbf{0}_{|O| \times N}$
$\mathfrak{M}_{\sigma^2} \leftarrow \mathbf{0}_{|O| \times N}$
$b_\mu \leftarrow \mathbf{0}_{|O| \times 1}$
$b_{\sigma^2} \leftarrow \mathbf{0}_{|O| \times 1}$
// Fill in the matrices
for all $O_{A_i} \in O$ **do**
  for all $a_j \in A_i$ **do**
    $\mathfrak{M}_\mu(i, j) \leftarrow \frac{1}{\binom{|A_i|}{2}} \sum_{\{a_j, a\} \in A_i} \phi(d(v_{a_j}, v_a))$
    $\mathfrak{M}_{\sigma^2}(i, j) \leftarrow \frac{1}{\binom{|A_i|}{2}^2} \sum_{\{a_j, a\} \in A_i} \phi(d(v_{a_j}, v_a))^2$
  end for
  $b_\mu(i) \leftarrow \frac{1}{|O_{A_i}|} \sum_{o \in O_{A_i}} o$
  $b_{\sigma^2}(i) \leftarrow \frac{1}{|O_{A_i}| - 1} \sum_{o \in O_{A_i}} (o - b_\mu(i))^2$
end for
// Solve least-squares for x where $\mathfrak{M}x = b$
means $\leftarrow$ `LeastSquares`$(\mathfrak{M}_\mu, b_\mu)$
variances $\leftarrow$ `LeastSquares`$(\mathfrak{M}_{\sigma^2}, b_{\sigma^2})$
for all $a_j \in \mathcal{A}$ **do**
  $C_{a_j} \sim \mathcal{N}(\text{means}(j), \text{variances}(j))$
end for
$C \leftarrow \bigcup_{a_j \in \mathcal{A}} C_{a_j}$
return C

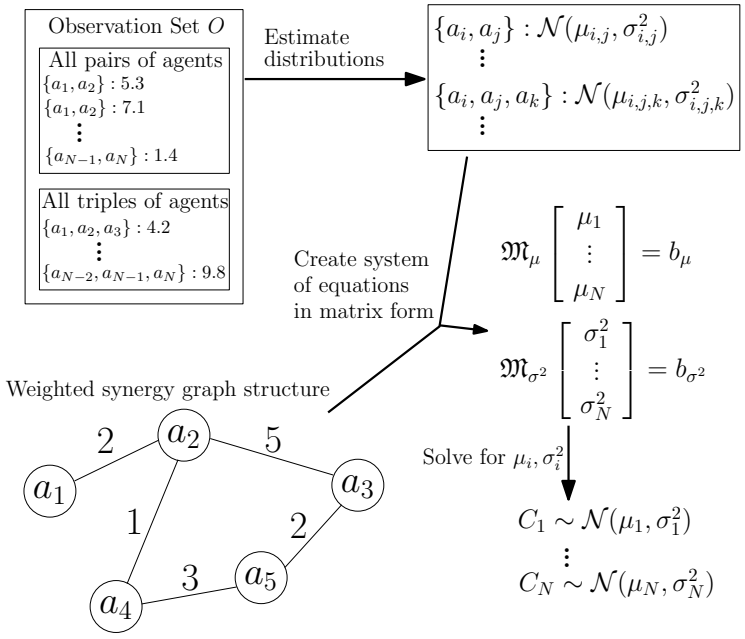


Figure 3.3: The capabilities of agents are learned from the observation set and a Weighted Synergy Graph structure using a least-squares solver.

Using Definition 2.4.2, the team $\{a_1, a_2\}$ forms the equation $\frac{1}{2}(C_1 + C_2)$. Furthermore, the observations involving $\{a_1, a_2\}$ in the observation set are used to estimate $\mathcal{N}(\mu_{\{a_1,a_2\}}, \sigma^2_{\{a_1,a_2\}})$, and two equations are generated:

$$\frac{1}{2}(\mu_1 + \mu_2) = \mu_{\{a_1,a_2\}}$$
$$\frac{1}{2^2}(\sigma_1^2 + \sigma_2^2) = \sigma^2_{\{a_1,a_2\}}$$

These two equations are separately formed and evaluated because the distributions $C_i \in C$ are independent. Since the distance between a_1 and a_2 is 2, $d_{i,j} = 2$ in Algorithm 5, and $\phi(d_{i,j}) = \frac{1}{2}$, which sets the values of columns 1 and 2 of $\mathfrak{M}_\mu$ to $\frac{1}{2}$ (Line 10). Similarly, Line 11 sets columns 1 and 2 of M_{σ^2} to be $\phi(d_{i,j})^2 = \frac{1}{2^2}$ using the second equation. The values of the corresponding rows of b_μ and b_{σ^2} are set to be $\mu_{\{a_1,a_2\}}$ and $\sigma^2_{\{a_1,a_2\}}$ respectively (Lines 13–14).

Once $\mathfrak{M}_\mu, \mathfrak{M}_{\sigma^2}, b_\mu$, and b_{σ^2} have been filled in by iterating through all teams with two and three agents and filling in the matrices, a least-squares solver is used to solve for the means and variances of the agent capabilities. These then form the agent capabilities $C = \{C_1, \ldots, C_N\}$ that are returned by the algorithm.

3.3.2 Learning Capabilities with a Non-Linear Solver

`LearnCapabilitiesNonLinear` learns the capabilities of the agents with a non-linear solver, using an existing Synergy Graph structure and the observation set O. There are N agents, and as such there are N Normal distributions to be estimated. We assume that $|O| > 2N$ so that the problem is over-constrained as opposed to underconstrained.

Similar to the previous subsection, the Synergy Graph structure is known, so the shortest distance between any two agents can be computed. `LearnCapabilitiesNonLinear` estimates Normal distributions so as to maximize the log-likelihood of the observations in O. From a single training example $(A, o_A) \in O$, an equation is formed that involves the means and variances of the agent capabilities.

We will use the example Weighted Synergy Graph structure in Figure 3.3. Suppose that the compatibility function is ϕ_{fraction}. The synergy $\mathbb{S}(\{a_1, a_2\}) = \phi(d(a_1, a_2))(C_1 + C_2)$, which simplifies to $\frac{1}{2}(C_1 + C_2)$. Hence, the log-likelihood of the example $o_{\{a_1,a_2\}}$ is:

$$-\frac{1}{2}\log(2\pi \cdot \frac{1}{4}(\sigma_1^2 + \sigma_2^2)) - \frac{(o_{\{a_1,a_2\}} - \frac{1}{2}(\mu_1 + \mu_2))^2}{2 \cdot \frac{1}{4}(\sigma_1^2 + \sigma_2^2)}$$

Thus, each observation in O corresponds to an expression involving the means and variances of the agent capabilities. In order to find the distributions that maximize the log-likelihood of O, the sum of log-likelihoods must be maximized. All log-likelihood expressions are summed and the non-linear solver is used to solve for the agent capabilities.

3.4 Computing Log-Likelihood and Accepting Neighbors

The Synergy Graph structure and learned capabilities are combined to form the Synergy Graph. The function `LogLikelihood` computes the sum of log-likelihood of every observation $o \in O$ given a Synergy Graph S. Every observation is o_A where $A \subseteq \mathcal{A}$ is an agent team, and $o_A \in \mathbb{R}$ is the observed performance of the team. Algorithm 6 calculates the log-likelihood of an observation set O, given a Synergy Graph S. In order to do so, for each observation group O_A in O, the synergy $\mathcal{N}(\mu_A, \sigma_A^2)$ of the group $A \subseteq \mathcal{A}$ is calculated using $\mathbb{S}$. Since the synergy is a Normal distribution, `LogLikelihood` computes the log-likelihood of each observed value in the observation group O_A, and sums the log-likelihoods across all the observations.

The score (i.e., summed log-likelihood of observations) of the current best Synergy Graph is compared to the score of the neighbor Synergy Graph, and accepted based on the approximation algorithm used. For example, when simulated annealing is used, the difference in the scores

Algorithm 6 Computing log-likelihood of an observation set

LogLikelihood(S, O)

1: score $\leftarrow 0$
2: **for all** $O_A \in O$ **do**
3: $C_A \leftarrow \mathbb{S}(A)$
4: **for all** $o \in O_A$ **do**
5: score $\leftarrow$ score $+ \log(\frac{1}{\sqrt{2\pi\sigma_A^2}} \exp(\frac{-(o-\mu_A)^2}{2\sigma_A^2}))$
6: **end for**
7: **end for**
8: **return** score

is compared to the temperature schedule. Hence, the Synergy Graph learning algorithm iteratively improves the Synergy Graph structure and learns the agent capabilities that best match the observations given.

3.5 Evaluating the Learning Algorithm

We describe the extensive experiments that show the efficacy of our learning algorithm. We first generate Synergy Graphs that are hidden from the learning algorithm. The observation set O used for learning is extracted from the hidden model, and then used to learn a new Synergy Graph, which we compare against the hidden one.

In order to quantitatively measure how well the learning algorithm performs, we used the log-likelihood of data given the hidden Synergy Graph. During training, the log-likelihood of the training examples is used to determine whether or not to accept a neighbor Synergy Graph. In addition, a set of test data is generated, that contains information that is never seen or used by the learning algorithm. We measured the log-likelihood of the test data given the learned Synergy Graph in each iteration of simulated annealing.

3.5.1 Learning Unweighted Synergy Graphs

In order to create random Unweighted Synergy Graphs, we first defined N, the number of agents, and EdgeProbability $\in (0, 1]$, the probability of an edge. In our experiments below, we iterated between values of EdgeProbability from $0.1, 0.2, \ldots, 0.9$ and collated the results across all EdgeProbability.

Then, the simulator generated the agents' capabilities. We generated $C_a \sim \mathcal{N}(\mu_a, \sigma_a^2)$ such that $\mu_a \in (-\gamma, \gamma)$ and $\sigma_a^2 \in (0, \gamma)$, where γ is a multiplying factor. γ affects how the performance of the agents are affected by the weight functions. For example, a compatibility of $\frac{1}{2}$ reduces a

capability of 4 to 2 (a difference of 2 units), but reduces 40 to 20 (a much larger difference of 20), which could have effects on the learning algorithm. Thus, we varied γ in our learning experiments to study the effect of the range of utilities on the performance of our algorithm.

We used 2 compatibility functions, ϕ_{fraction} (Equation 2.1) and ϕ_{decay} (Equation 2.2). The two compatibility functions were selected to demonstrate that the algorithms' performance is similar regardless of the compatibility function, and because both ϕ_{fraction} and ϕ_{decay} were intuitive and easy to understand. For these experiments, we set the half-life $h = 3$ in ϕ_{decay}, since $|\mathcal{A}|$ was set to be at most 10, so the compatibility between agents would have a similar range for both functions.

Log-Likelihoods of Learned Unweighted Synergy Graphs

We first generated an Unweighted Synergy Graph S_{hidden} using the method described above, and then generated a training observation set O_{train}, with sets of 2 and 3 agents, i.e., $\forall\, O_A \in O_{train}$, $2 \leq |A| \leq 3$. We generated 100 data points from each pair/triple, and learned a Synergy Graph $S_{learned}$ from the data.

Then, to test how well our algorithm learns the Unweighted Synergy Graph, we generated a test observation set O_{test} using combinations of 4 or greater agents, i.e., $\forall\, O_A \in O_{test}, |A| \geq 4$, that had 1000 observations in total. We then measured the difference in log-likelihoods between the hidden and learned Unweighted Synergy Graphs, i.e., $LL(O_{test}|S_{true}) - LL(O_{test}|S_{learned})$, where $LL(O|S) = \texttt{LogLikelihood}(O, S)$. A low log-likelihood difference indicates that the Unweighted Synergy Graph is as likely as the true graph to have produced the observations. We compared this difference in log-likelihood versus the initial random Synergy Graph used in the learning algorithm, $S_{initial}$, that had random edges but learned agent capabilities, to observe if the graph structure has an effect on the log-likelihoods.

Figure 3.4 shows the log-likelihood differences of the learned Unweighted Synergy Graphs with the 2 compatibility functions compared to the hidden Unweighted Synergy Graph, and varying the number of agents from 6 to 10. Varying the number of agents does not affect the log-likelihood error much — the compatibility function and the multiplier γ have greater effects.

Figures 3.5a and 3.5b show the log-likelihood difference of the learned Unweighted Synergy Graphs and the initial Unweighted Synergy Graphs when $|\mathcal{A}| = 10$. The learned Unweighted Synergy Graphs are much closer to the hidden Unweighted Synergy Graphs, i.e., the difference in log-likelihood is close to 0 and orders of magnitude lower than the initial Synergy Graphs. The error in log-likelihood increases as γ, the multiplying factor, increases, especially for $S_{initial}$, and shows that γ affects the difficulty of the learning problem (seen from the errors of $S_{initial}$), but our learning algorithm is capable of significantly reducing this error in $S_{learned}$. Furthermore, the

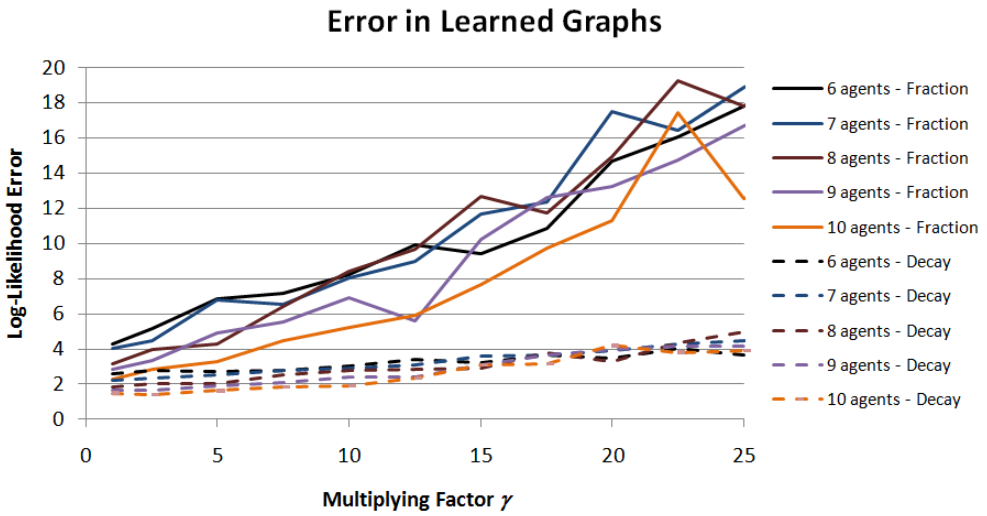
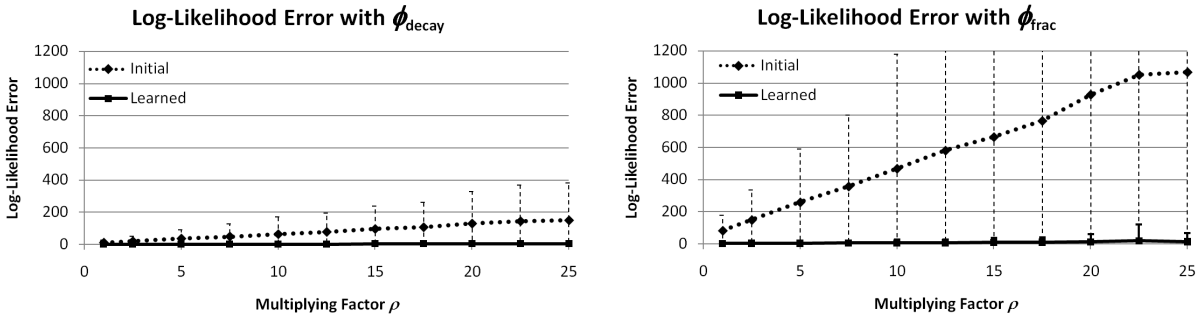


Figure 3.4: The error in the learned Unweighted Synergy Graph with varying number of agents and both compatibility functions.



(a) (b)

Figure 3.5: The error in the learned Unweighted Synergy Graph of 10 agents with heterogeneous task performance, using the compatibility function a) $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{3})$ and b) $\phi_{\text{fraction}}(d) = \frac{1}{d}$, compared with the initial Unweighted Synergy Graph used by the learning algorithm, with random structure but learned agent capabilities.

observation set used for learning only included pairs and triples of agents, but the learned graph had a low log-likelihood difference when testing against data of teams comprising 4 or more agents, which shows that the structure of the learned graph and the individual agent capabilities match the hidden Synergy Graph well.

Measuring Team Performance

The goal is to find the optimal multi-agent team, and we use `Approx`δ`OptimalTeam` (Algorithm 2) on the learned Synergy Graphs $S_{learned}$. The performance of this set of agents is then computed in the hidden Unweighted Synergy Graph S_{hidden}, and compared against the best and

	Learned Graph	Hidden Graph
6 agents	0.9990 ± 0.0073	1.0000 ± 0
7 agents	0.9989 ± 0.0045	1.0000 ± 0.0005
8 agents	0.9994 ± 0.0032	1.0000 ± 0
9 agents	0.9996 ± 0.0028	1.0000 ± 0
10 agents	0.9995 ± 0.0036	0.9999 ± 0.0012

Table 3.1: Effectiveness of multi-agent teams formed.

worst possible combinations of agents. For example, if the set of agents A' is selected from $S_{learned}$, then the value of A' is computed on S_{hidden}.

We varied $|\mathcal{A}|$, the number of agents in the Synergy Graph, from 6 to 10, and the algorithm picked the best 5 agents. We set $\gamma = 1$, and $\rho = 0.75$. Table 3.1 show the effectiveness of the picked teams (Equation 2.6), where 0 denotes the worst possible combination, and 1 is the optimal team, and the performance of the selected team on the hidden Unweighted Synergy Graph S_{hidden}. The worst and optimal teams were discovered by iterating through all possible combinations of agents and computing their value. It is remarkable that our algorithm finds a team that obtains a score of at least 0.9989, and has a similar score when the algorithm is run on the hidden Unweighted Synergy Graph, and thus shows that the learned Unweighted Synergy Graphs are in fact very close to hidden Unweighted Synergy Graphs that were used to generate the observation sets, and that `Approx`δ`OptimalTeam` can be used to find effective teams.

3.5.2 Learning Representative Weighted Graph Structures

In this set of experiments, we used hidden Weighted Synergy Graphs that were of three representative structure types — chain, loop, and star. The learning algorithm does not know the structure type of the hidden Weighted Synergy Graph, and instead starts with a random graph structure. Figure 3.6 shows the hidden, initial and final Weighted Synergy Graphs for each of the representative structure types. When the hidden Synergy Graphs are generated, the edge weights are random integers in the range $[1, 10]$. The limits were chosen to provide reasonable bounds for the learning algorithm's exploration of structures, while allowing for a significant difference in compatibility among agents.

It is very interesting that the structure learned closely matches the hidden Synergy Graph, even though the learning algorithm has no prior regarding such structures. The algorithm randomly generates a graph by creating edges between all possible pairs of vertices with 50% probability (i.e., EdgeProbability $= 0.5$ in Algorithm 4).

Since each trial uses a randomly generated hidden graph, log-likelihood values vary from

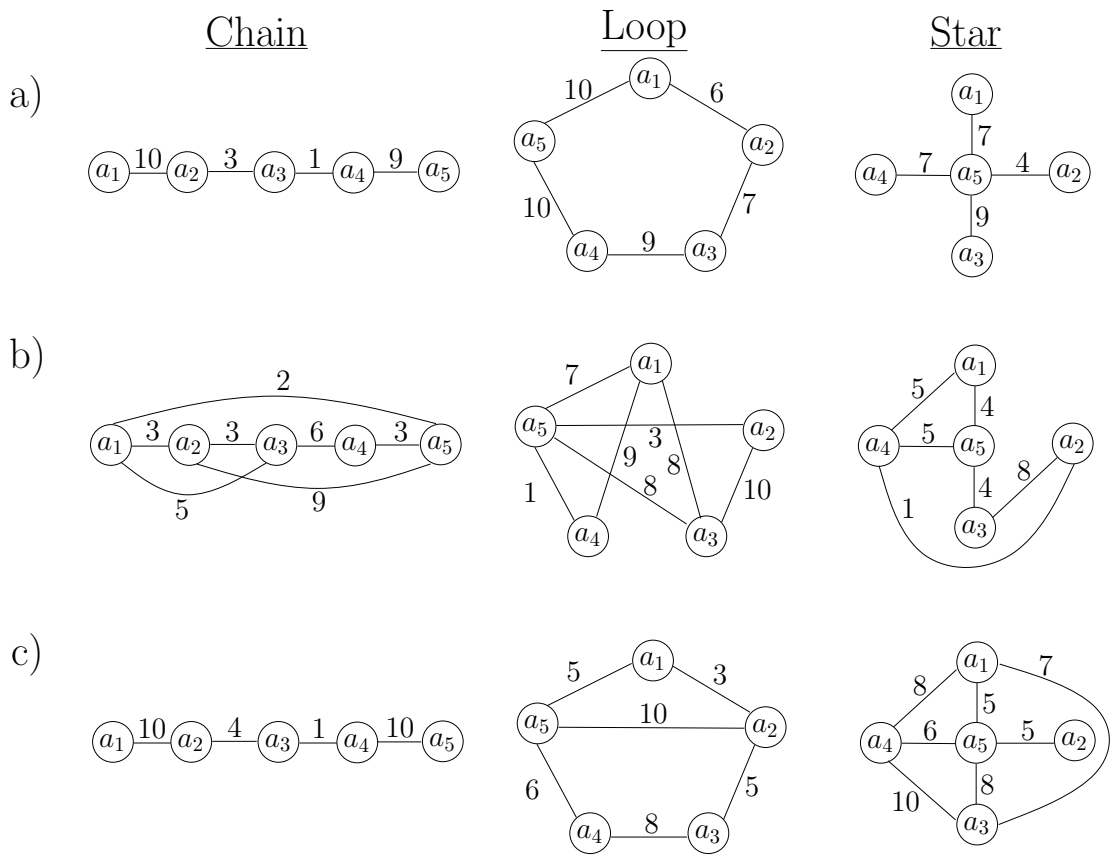


Figure 3.6: Results of the learning algorithm using representative graph structure types. The agent capabilities are not shown, and the vertices are laid out for visual purposes. a) Examples of Weighted Synergy Graphs with 5 agents generated to form representative structure types. b) The initial randomly-generated Weighted Synergy Graph of the learning algorithm. c) Learned Weighted Synergy Graphs corresponding to the Weighted Synergy Graphs in *(a)*, after 1000 iterations of simulated annealing.

trial to trial. Thus, we scaled the log-likelihood numbers to be between 0 and 1, where 0 is the log-likelihood of the initial guess, and 1 means that the log-likelihood of the learned Weighted Synergy Graph matches that of the hidden one used to generate the data. Figure 3.7 shows the learning curves of simulated annealing for 1000 iterations with 10 agents and averaged over 100 trials per structure type. While the scaled log-likelihood of *star* rises very quickly compared to *loop* and *chain*, the latter two outperform *star* after 600 iterations. The final score of *star*, *loop*, and *chain* are 0.93, 0.95 and 0.96 respectively, which shows that the learned Weighted Synergy Graphs closely matches the hidden ones. The experiments were run in Java using MATLAB as the least-squares solver, on a Intel Quad-Core 2.4 GHz machine. On average, the learning algorithm took 20.6 ± 0.3 seconds to perform 1000 iterations of simulated annealing to learn the Weighted Synergy Graph with 10 agents, and was consistent across the structure types.

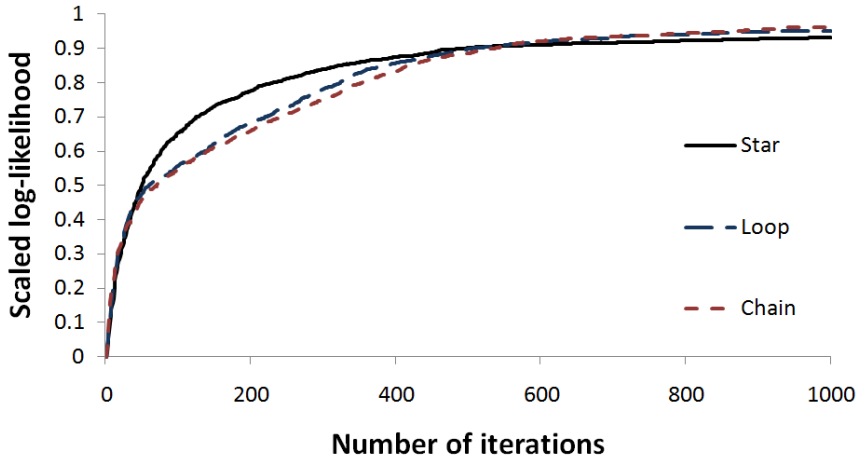


Figure 3.7: Performance of the learning algorithm with different Weighted Synergy Graph structure types, averaged over 100 trials.

3.5.3 Learning Random Weighted Synergy Graphs

We randomly generated Weighted Synergy Graphs, to further explore the space of possible graph structures. We varied the number of agents from 10 to 15, and varied the agent capabilities C with a scale-factor γ. Each agent capability $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ was randomly generated such that $\mu_i \in (\frac{\gamma}{2}, \frac{3\gamma}{2})$ and $\sigma_i^2 \in (0, \gamma^2)$. Thus, a higher value of γ creates a larger range of possible agent capabilities, and we wanted to investigate if it would have any effect on the learning algorithm.

Figure 3.8 shows the performance of the learning algorithm with a varying number of agents, and $\gamma = 10$, averaged over 100 trials per number of agents. The shape of the learning curve is consistent, and the performance of the learning algorithm decreases slightly as the number of agents increases. Table 3.2 shows the final scaled log-likelihood of the learned Weighted Synergy Graph as the number of agents and scale-factor γ varies. In all cases, the scaled log-likelihood is 0.89 or higher, which indicates that the learned Weighted Synergy Graph closely matches the hidden one. The scale-factor γ has little to no effect of the final learning outcome, while an increase in the number of agents decreases the score slightly.

In the experiments above, we used the compatibility function ϕ_{fraction}. Figure 3.9 shows the learning curves of the learning algorithm with the compatibility functions ϕ_{fraction} and ϕ_{decay}. Although ϕ_{fraction} increases more slowly than ϕ_{decay}, the final score of the algorithm is similar after all 1000 iterations, which indicates that while the compatibility function has an effect on the learning rate, the final outcome is similar.

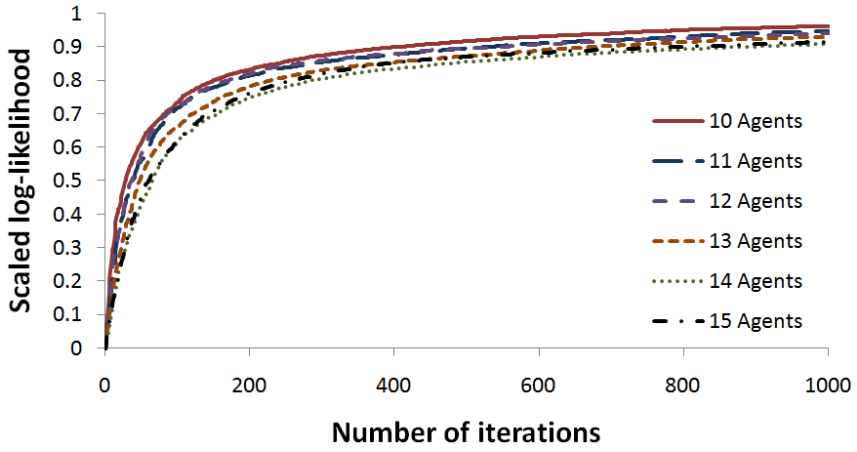


Figure 3.8: Performance of the learning algorithm on random weighted graph structures with varying number of agents, averaged over 100 trials.

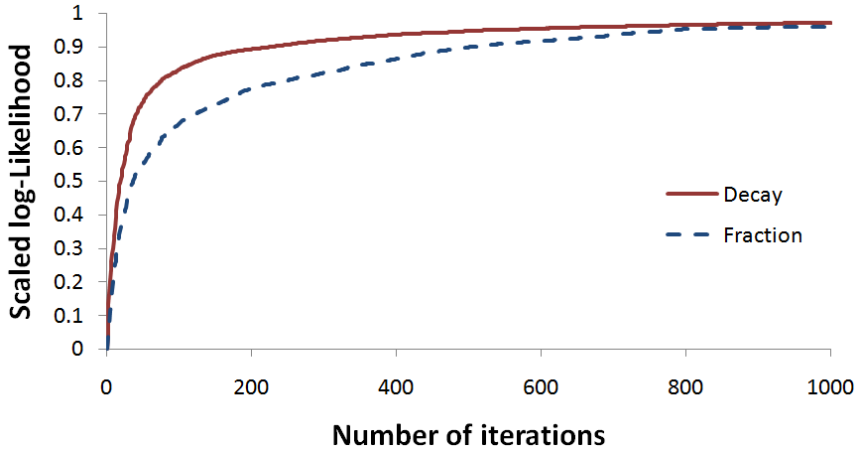


Figure 3.9: Learning curves of two compatibility functions — ϕ_{decay} and ϕ_{fraction}, averaged over 100 trials.

# agents	Scale-factor of agent capabilities (γ)				
	1.0	2.5	5.0	7.5	10.0
10	0.96	0.95	0.97	0.97	0.96
11	0.96	0.95	0.95	0.95	0.95
12	0.93	0.93	0.94	0.93	0.94
13	0.93	0.93	0.93	0.92	0.93
14	0.90	0.93	0.93	0.92	0.91
15	0.89	0.90	0.91	0.91	0.91

Table 3.2: Scaled log-likelihood of the learned Weighted Synergy Graphs, varying the number of agents, and γ, the scale-factor of agent capabilities.

3.5.4 Comparing the Capability Learning Algorithms

The experiments above learned Unweighted and Weighted Synergy Graphs using `LearnCapabilityLeastSquares`, the agent capability learner that uses a least-squares solver. In the experiments below, we evaluate the performance of using the non-linear solver to learn agent capabilities with `LearnCapabilityNonLinear`. In all the experiments below, we use Weighted Synergy Graphs, and we believe the results are representative of Unweighted Synergy Graphs as well.

First, we varied the number of agents N from 10 to 15, and generated 100 hidden Weighted Synergy Graphs S_{hidden} with $\gamma = 100$ for each value of N. We generated the training observation set O_{all} by retrieving 15 samples of performance for each agent pair and triple. Next, we generated subsets of O_{all} by using up to one sample per agent pair and triple, and limiting the total number of agent teams to 50 and 100, forming the training observations sets O_{50} and O_{100} respectively. Using the graph structure of the hidden Weighted Synergy Graph and O_{all}, `LearnCapabilityLeastSquares` learned a Weighted Synergy Graph S_{ls}, and O_{all}, O_{50}, and O_{100} were used with `LearnCapabilitiesNonLinear` to learn S_{nl}, S_{50}, and S_{100} respectively. To evaluate the learned Weighted Synergy Graphs, we generated the test observation set O_{test} by retrieving 30 samples of performance for every agent team that had greater than 3 members. The log-likelihood of O_{test} given the learned Weighted Synergy Graph was then computed.

Figure 3.10 shows the log-likelihoods of the learned Weighted Synergy Graphs, compared to the log-likelihood of the hidden Weighted Synergy Graph. The log-likelihood of S_{ls} is close to that of S_{hidden}, showing that the least-squares solver is able to effectively learn the agent capabilities. The log-likelihood of S_{nl} is slightly above half of S_{ls} even though both used the same observation set, reflecting that the non-linear solver generally does more poorly than the least-squares solver. There are two main reasons for the discrepancy. First, the least-squares solver solves the means and variances of the agent capabilities separately. Second, the least-squares solver uses the synergy equations to compute the means and variances directly, while the non-linear solver has to solve the actual log-likelihood equation that involves the Normal distribution probability density function.

Although the non-linear solver does not perform as well as the least-squares solver, it has a major advantage: it can learn agent capabilities even when the observation set is small. The log-likelihoods of S_{50} and S_{100} are considerably lower than S_{ls} and S_{nl}, but they only used 50 and 100 observations respectively, compared to $15 \cdot (\binom{N}{2} + \binom{N}{3})$ for S_{ls} and S_{nl}. For example, where $N = 15$, S_{ls} and S_{nl} had 8400 observations. Hence, the non-linear solver provides a method to learn agent capabilities with varying number of observations, with a trade-off between accuracy and observation size.

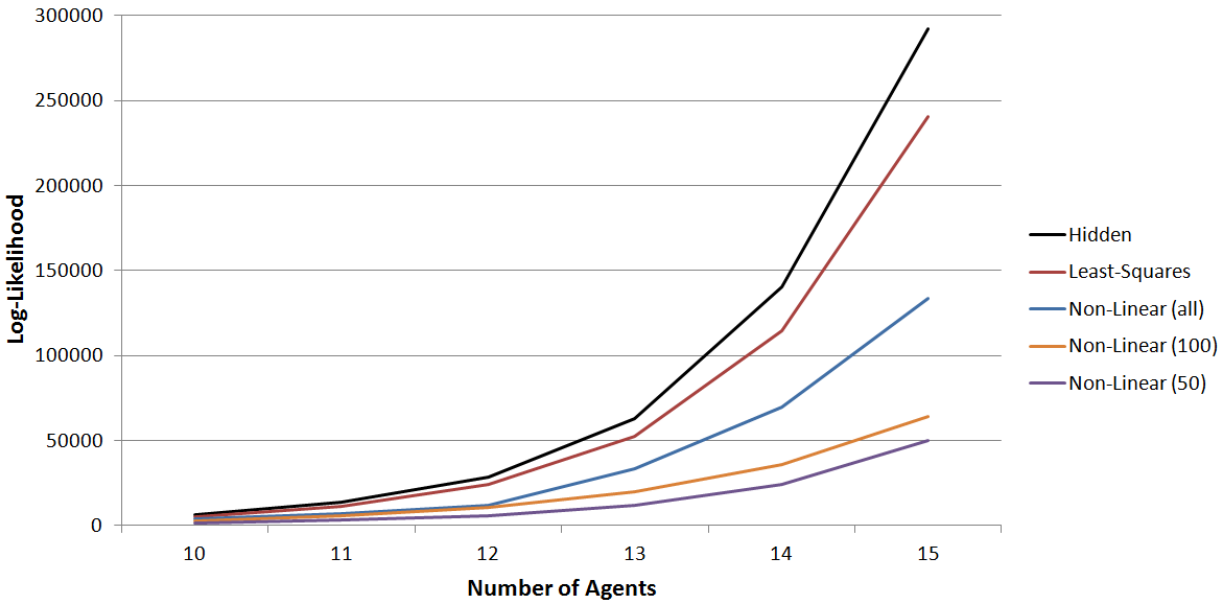


Figure 3.10: Log-likelihoods of learned Weighted Synergy Graphs using a least-squares solver and non-linear solver to learn the agent capabilities, compared to the log-likelihood of the hidden Weighted Synergy Graph.

3.6 Chapter Summary

This chapter presented `LearnSynergyGraph`, the Synergy Graph learning algorithm that iteratively improves the Synergy Graph structure and learns the agent capabilities. The only input to the algorithm is a set of observations of team performance, and the algorithm begins by generating a random Synergy Graph structure and learning the agent capabilities using the generated structure and observation set. Next, it enters an approximation loop, where it iteratively modifies the Synergy Graph structure and re-learns the agent capabilities, and accepts the new Synergy Graph candidate based on the approximation algorithm (i.e., based on the difference in log-likelihoods and temperature schedule in simulated annealing). This chapter presented `LearnCapabilitiesLeastSquares` and `LearnCapabilitiesNonLinear`, that learn the agent capabilities using an existing Synergy Graph structure and the given observation set. The former algorithm uses a least-squares solver, while the latter uses a non-linear solver. The efficacy of the learning algorithm was demonstrated with a variety of experiments on Unweighted Synergy Graphs and Weighted Synergy Graphs.

Chapter 4

Iteratively Learning a New Teammate

The Synergy Graph model is learned from data, with the underlying assumption that data about all the agents are available initially. This chapter focuses on how to incorporate new information into the model, and introduces the `AddTeammateToSynergyGraph` algorithm, that uses observations of a new agent working with the previous agents, to add the new agent into an existing Synergy Graph [Liemhetcharat and Veloso, 2013b]. This chapter presents three heuristics for generating the edges that connect the new agent to the previous agents, and experiments that compare these heuristics. This chapter also compares different learning approaches: completely re-learning the SynergyGraph with `LearnSynergyGraph`, learning all but one agent with `LearnSynergyGraph` followed by one agent with `AddTeammateToSynergyGraph`, and learning all the agents incrementally with `AddTeammateToSynergyGraph`.

4.1 Learning Algorithm for Adding a Teammate

Recall that $\mathcal{A} = \{a_1, \ldots, a_N\}$ is the set of agents initially known. The learning algorithm `LearnSynergyGraph` assumes that the observation set O about teams in $\mathcal{A}$ is initially available, and the algorithm learns a Synergy Graph model S from O.

Suppose there is a new agent a_{N+1}, and let the new set of agents be $\mathcal{A}^+ = \mathcal{A} \cup \{a_{N+1}\}$. Also suppose that observations about the new agent are available, and let O_{N+1} be the set of new observations. We will elaborate on the details of O_{N+1} later. The goal of this chapter is to learn an updated synergy graph S^+ that models the agents $\mathcal{A}^+$.

`LearnSynergyGraph` is capable of learning a synergy graph to model $\mathcal{A}^+$: by using the union of all the observations $O^+ = O \cup O_{N+1}$. However, using O^+ to learn a new model has two main drawbacks. First, the previously learned Synergy Graph S is discarded. Second, the runtime is high — the model of all $N + 1$ agents and their capabilities have to be learned.

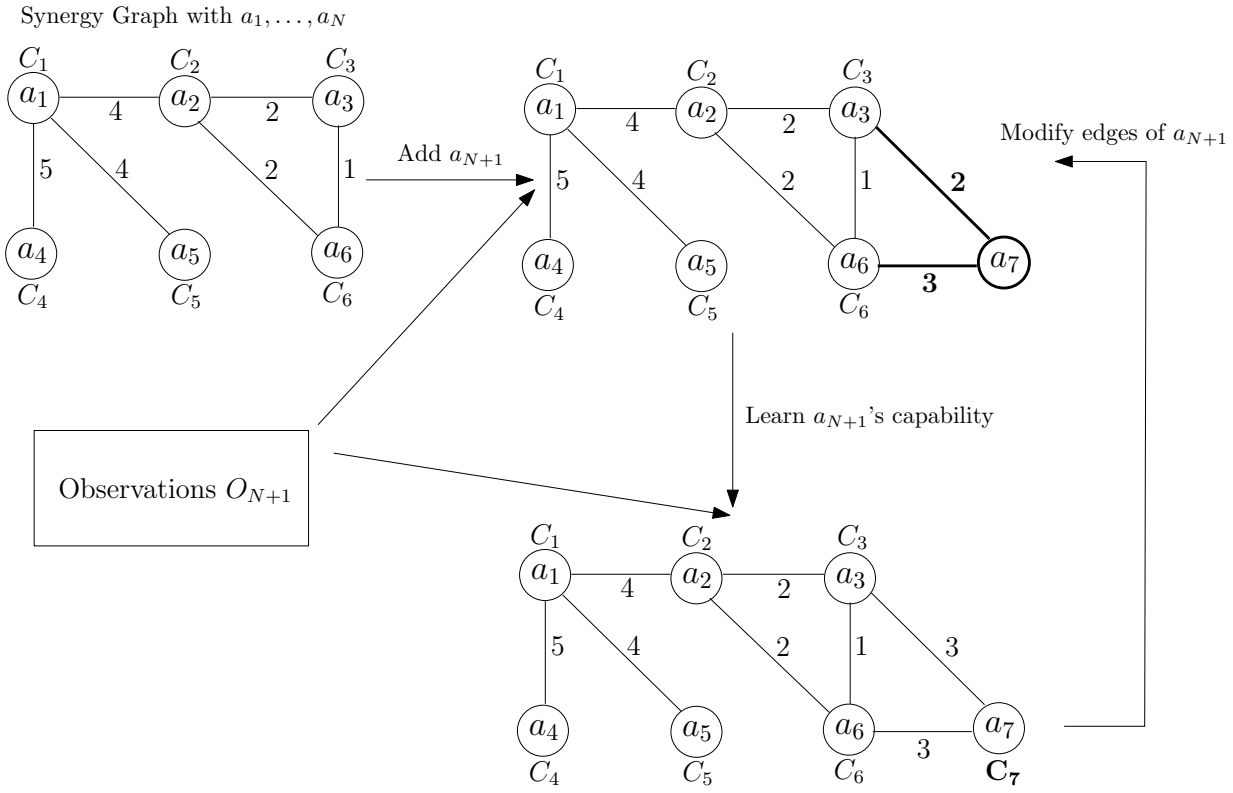


Figure 4.1: The learning algorithm `AddTeammateToSynergyGraph` adds a new teammate into a Synergy Graph (a Weighted Synergy Graph is used in this figure). Simulated annealing is performed, where edges of a_{N+1} are modified, and the capability C_{N+1} is learned.

Our approach is to use the previously learned synergy graph S to jump-start the learning. We assume that S models the interactions and capabilities of the agents in $\mathcal{A}$, and only seek to learn a_{N+1}'s capability, and its interactions with the other agents, i.e., the weighted edges that connect a_{N+1} to the other agents. While we consider a new agent a_{N+1}, our approach can be applied to re-learning an existing agent's capabilities and synergy, i.e., $a_i \in \{a_1, \ldots, a_N\}$. For example, the agent a_i has a changed capability due to a broken sensor since the original observations were made. Re-learning a_i's capability and synergy is done by setting $A' = \{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N\}$ and $a'_{N+1} = a_i$. In such a situation, The existing Synergy Graph S would only contain the agents in A' — if removing a_i's edges disconnects the graph, edges can be added with weights equal to the shortest distance between vertices before removing a_i.

Figure 4.1 shows our learning process, and Algorithm 7 shows the pseudo-code. The key difference in `AddTeammateToSynergyGraph` versus `LearnSynergyGraph` is that the former only considers edges that connect a_{N+1} to other agents in $\mathcal{A}$, and only learns a_{N+1}'s capability.

Algorithm 7 Add a new teammate to a Synergy Graph

AddTeammateToSynergyGraph(S, a_{N+1}, O_{N+1})

// Add a new vertex to represent a_{N+1}
$V^+ \leftarrow V \cup a_{N+1}$
// Generate initial edges to connect a_{N+1} to $\mathcal{A}$
$E_{\text{initial}} \leftarrow \texttt{GenerateTeammateEdges}(S, a_{N+1}, O_{N+1})$
$E^+ \leftarrow E \cup E_{\text{initial}}$
// Form the new Synergy Graph structure
$G^+ \leftarrow (V^+, E^+)$
// Learn a_{N+1}'s capability
$C_{N+1} \leftarrow \texttt{LearnTeammateCapability}(G^+, O_{N+1})$
// Form the initial Synergy Graph
$S^+ \leftarrow (G^+, C \cup C_{N+1})$
score $\leftarrow \texttt{LogLikelihood}(S^+, O_{N+1})$
// Simulated annealing loop
for $k = 1$ **to** $k_{\max}$ **do**
  $G' \leftarrow \texttt{NeighborTeammateEdges}(G^+, a_{N+1})$
  $C'_{N+1} \leftarrow \texttt{LearnTeammateCapability}(G', O_{N+1})$
  $S' \leftarrow (G', C \cup C'_{N+1})$
  score′ $\leftarrow \texttt{LogLikelihood}(S', O_{N+1})$
  if $\mathbb{P}(\text{score}, \text{score}', \text{temp}(k, k_{\max})) > \texttt{random}()$ **then**
    $S^+ \leftarrow S'$
    score $\leftarrow$ score′
  end if
end for
return S^+

There are three main components to `AddTeammateToSynergyGraph`: generating the initial edges (`GenerateTeammateEdges`), generating neighbor edges in the simulated annealing loop (`NeighborTeammateEdges`), and learning a_{N+1}'s capability (`LearnTeammateCapability`) given the structure of the graph. We will explain each of these three functions in detail next.

4.2 Generating the Teammate's Initial Edges

The goal of the `GenerateTeammateEdges` function (Line 4 of Algorithm 7) is to generate edges that connect the new vertex a_{N+1} to the other vertices $(a_1, \ldots, a_N)$ in the Synergy Graph. Recall that a Synergy Graph is a connected graph, and so a_{N+1} must have at least one edge that connects it to another vertex.

One method is to randomly generate these edges. Algorithm 8 shows how this is done. A probability $0 < p \leq 1$ is defined, and for every possible edge connecting a_{N+1} with another vertex, a dice is thrown and the edge is created with probability p. If the edge is weighted, then the weight of the created edge is also randomly chosen to be an integer between w_{min} and w_{max}. The while-loop ensures that at least one edge is created so a_{N+1} is connected to some other vertex.

While `GenerateRandomTeammateEdges` suffices to create an initial guess, it requires a defined probability p, which may be domain-specific and difficult to ascertain. An improved method, `GenerateTeammateEdgesWithDensity` (Algorithm 9), first estimates p by examining the existing Synergy Graph S and determining the density of edges (i.e., number of edges divided by the number of possible edges) in that graph. `GenerateTeammateEdgesWithDensity` makes the following assumption:

Assumption 4.2.1. *The number of edges connecting the new agent a_{N+1} with other agents is similar to the edge density of S.*

Line 2 of Algorithm 9 only counts edges with two unique end-points, so that self-looping edges are not counted (which are present in some Synergy Graphs such as the WeSGRA model described in Chapter 5).

Algorithm 8 Generate edges with random probability p

GenerateRandomTeammateEdges(S, a_{N+1}, p)

$E_{\text{new}} \leftarrow \emptyset$
while $E_{\text{new}} = \emptyset$ **do**
  $E_{\text{new}} \leftarrow \emptyset$
  for all $a \in \{a_1, \ldots, a_N\}$ **do**
    if $p >$ random() **then**
      if `isUnweightedSynergyGraph`() **then**
        $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{(a, a_{N+1})\}$
      else if `isWeightedSynergyGraph`() **or** `isWeSGRA`() **then**
        $w \leftarrow \texttt{randint}(w_{\text{min}}, w_{\text{max}})$
        $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{(a, a_{N+1}, w)\}$
      else if `isSGRaCR`() **or** `is`ρ`-SGraCR`() **then**
        $w_{\text{intra}} \leftarrow \texttt{randint}(w_{\text{min}}, w_{\text{max}})$
        $w_{\text{inter}} \leftarrow \texttt{randint}(w_{\text{min}}, w_{\text{max}})$
        $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{(a, a_{N+1}, w_{\text{intra}}, w_{\text{inter}})\}$
      end if
    end if
  end for
end while
if `isWeSGRA`() **or** `isSGRaCR`() **or** `is`ρ`-SGraCR` **then**
  $w \leftarrow \texttt{randint}(w_{\text{min}}, w_{\text{max}})$
  $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{a_{N+1}, a_{N+1}, w\}$
end if
return E_{new}

Algorithm 9 Determine density of other edges and generate new edges

GenerateTeammateEdgesWithDensity(S, a_{N+1})

// $S = ((V, E), C)$
$n' \leftarrow |\{(v_i, v_j, w_{i,j}) \in E \mid v_i \neq v_j\}|$
density $\leftarrow \frac{n'}{\binom{N}{2}}$
$E_{\text{new}} \leftarrow$ `GenerateRandomTeammateEdges`$(S, a_{N+1},$ density$)$
return E_{new}

The third method to generate the initial edges also uses the existing edges of the synergy graph S. However, instead of computing the density of edges, `GenerateSimilarTeammateEdges` (Algorithm 10) finds the agent $a_i \in \mathcal{A}$ most similar to a_{N+1}, and duplicates all its edges. This method is similar to the *nearest-neighbor* heuristic in Machine Learning classification, but the difference is that we are interested in duplicating the edges of the nearest neighbor. The similarity between agents is computed using the observations in O_{N+1}, which contain observations of teams containing a_{N+1}. For example, suppose that one observation $o \in O_{N+1}$ is (A', v), which indicates a team $\{a_{N+1}, a_j\} = A' \subseteq \mathcal{A}$ that had a performance of v. A new synthetic observation $o' = (\{a_i, a_j\}, v)$ is created where a_{N+1} is replaced with a_i. All such synthetic observations form a new set O', and the log-likelihood of O' given S is computed. The most similar agent to a_{N+1} is then the one with the highest log-likelihood. `GenerateSimilarTeammateEdges` makes the following assumption:

Assumption 4.2.2. *The new agent a_{N+1} is similar to another agent a_i already present in the Synergy Graph S, and hence has similar edges as a_i.*

Algorithm 10 Replicate edges of most similar agent

GenerateSimilarTeammateEdges(S, a_{N+1}, O_{N+1})

$a_{\text{closest}} \leftarrow a_1$
score $\leftarrow -\infty$
for all $a_i \in \mathcal{A}$ **do**
  $O' \leftarrow \emptyset$
  for all $o = (A', v) \in O_{N+1}$ **s.t.** $a_i \notin A'$ **do**
    $O' \leftarrow O' \cup \{(A' \setminus \{a_{N+1}\} \cup \{a_i\}, v)\}$
  end for
  score$'$ $\leftarrow$ `LogLikelihood`(S, O')
  if score$'$ $>$ score **then**
    $a_{\text{closest}} \leftarrow a_i$
    score $\leftarrow$ score$'$
  end if
end for
$E_{\text{new}} \leftarrow \emptyset$
for all $e = (a_i, a_j, w_{i,j}) \in E$ **s.t.** $a_i = a_{\text{closest}}$ **do**
  $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{(a_{N+1}, a_j, w_{i,j})\}$
end for
return E_{new}

4.3 Generating Neighbor Edges

The three functions (`GenerateRandomTeammateEdges`, `GenerateTeammateEdgesWithDensity`, and `GenerateSimilarTeammateEdges`) described previously create an initial guess of the edges connecting a_{N+1} to the other vertices in the Synergy. During the simulated annealing search, new candidate Synergy Graph structures are generated, so as to effectively explore the space of all possible edges.

We use the same four actions of neighbor generation as `NeighborStructure` in `LearnSynergyGraph` (Algorithm 3), except that we only consider edges involving a_{N+1}, i.e., an edge $e = (a_{N+1}, a_i, w_{i,j})$:

- Remove an existing edge if it does not disconnect a_{N+1},
- Add a new edge with a randomly-generated weight w,
- Increase the weight of an edge by 1, subject to w_{max},
- Decrease the weight of an edge by 1, subject to w_{min}.

There are $(w_{\text{max}} - w_{\text{min}} + 1)^N$ possible edges that connect a_{N+1} to the other vertices, and so it is infeasible to consider all combinations of edges. Through these four actions, we can explore the space of such edges iteratively. Further, since only edges involving a_{N+1} are considered, a much smaller and restricted space of edges are considered compared to all possible edges in `NeighborStructure`. Thus, the space is better explored with the same number of simulated annealing iterations. We compare the effectiveness of searching this restricted space versus the entire space of edges later.

4.4 Learning the Teammate's Capability

The Synergy Graph learning algorithm learns the capabilities of all the agents in the synergy graph, using two algorithms: `LearnCapabilitiesLeastSquares` and `LearnCapabilitiesNonLinear`. The former requires that O (the set of observations involving $a_1, \ldots, a_N$) contains all teams of size 2 and 3, with multiple observations per team. A Normal distribution per observed team is estimated using the data, and a matrix least-squares solver is used to solve for the means and variances of the agent capabilities. In `LearnCapabilitiesNonLinear`, O contains samples of team performance, and each observation in O forms a log-likelihood expression involving the means and variances of the agents in the team. A non-linear solver then solves for the agent capabilities using the expressions to maximize the log-likelihood.

The `LearnTeammateCapability` algorithm learns agent a_{N+1}'s capability using the Synergy Graph structure and the observation set O_{N+1}. We assume that the capabilities of

$a_1, \ldots, a_N$ are known, so the only unknowns are μ_{N+1} and σ^2_{N+1}, the mean and variance of a_{N+1}'s capability, i.e., $C_{N+1} \sim \mathcal{N}(\mu_{N+1}, \sigma^2_{N+1})$.

We use two techniques to learn a_{N+1}'s capability, which are modified from `LearnCapabilitiesLeastSquares` and `LearnCapabilitiesNonLinear`. If O_{N+1} contains observations of all teams of size 2 and 3 involving a_{N+1}, i.e., $\forall o = (t, v) \in O_{N+1}, a_{N+1} \in t$, then we use a matrix least-squares operation to solve for C_{N+1}. Otherwise, we form log-likelihood expressions and use a non-linear solver.

4.5 Analyzing the Iterative Learning Algorithm

We contributed three functions for generating the edges of the Synergy Graph to initialize the learning algorithm to include agent a_{N+1} into the Synergy Graph:

- `GenerateRandomTeammateEdges`, that generates random edges with a given probability $0 < p \leq 1$;
- `GenerateTeammateEdgesWithDensity`, that generates edges randomly based on the density of edges in the Synergy Graph containing $a_1, \ldots, a_N$;
- `GenerateSimilarTeammateEdges`, that replicates the edges of the most similar agent to a_{N+1}.

We also contributed two functions to learn the agent a_{N+1}'s capability, using a matrix least-squares computation, and using a non-linear solver. In this section, we compare the three edge generation functions and two capability learning functions to analyze their characteristics and performance.

4.5.1 Experimental Setup

Figure 4.2 shows the process of our experiments. We first randomly generate a Weighted Synergy Graph S^* with $N + 1$ agents, and label the vertices v_1 to v_{N+1}, such that $\forall 1 \leq i \leq N + 1$, the subgraph with vertices $v_1, \ldots, v_i$ remains connected. In particular, the subgraph containing $v_1, \ldots, v_N$ forms the pre-existing Weighted Synergy Graph to our learning algorithm (S in the input to Algorithm 7). Figure 4.3 shows the four different graph structures of Weighted Synergy Graphs created: chain, loop, star, and random structure (where edges are added probabilistically). For each graph structure type, we generate 50 Weighted Synergy Graphs, and hence 200 Weighted Synergy Graphs S^* are generated in total.

From the Weighted Synergy Graph S^*, we generate the observation set O_{N+1}. For the experiments using the matrix least-squares agent capability learner, we use all pairs and triples of

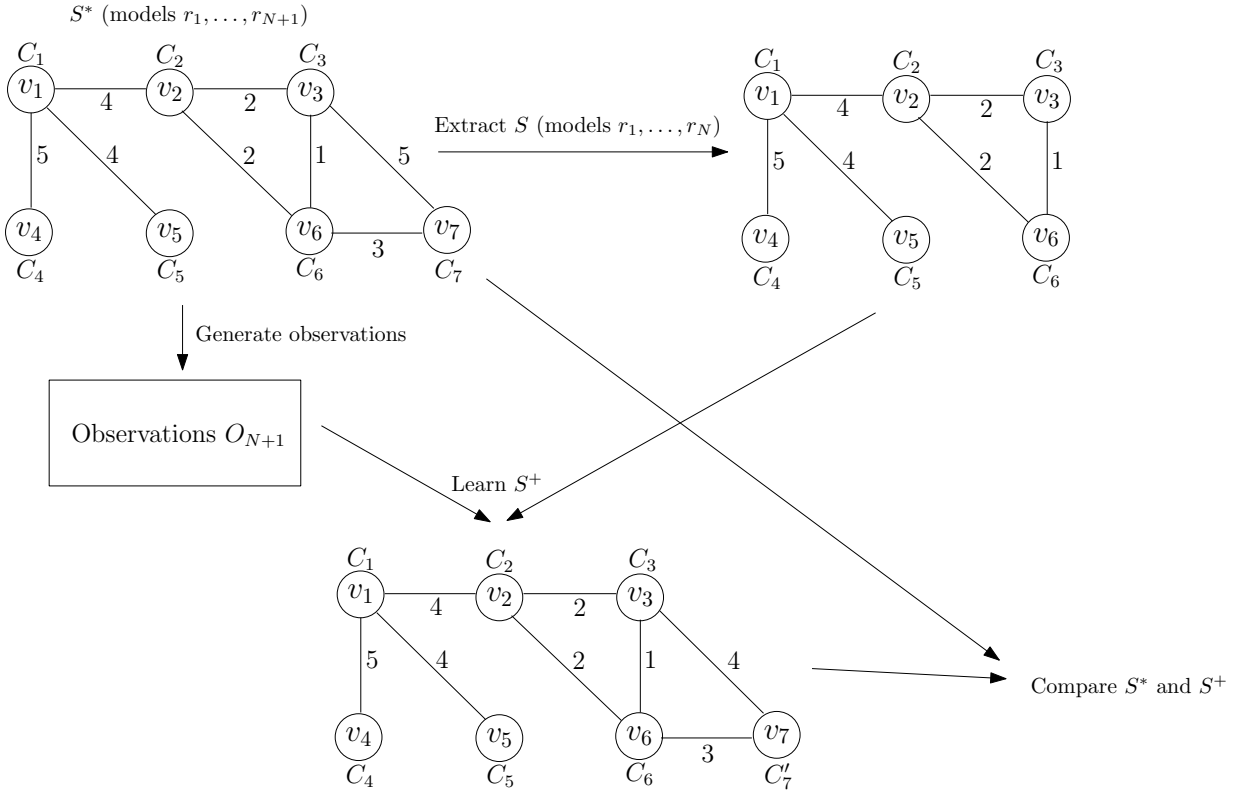


Figure 4.2: The experimental process to compare the initial edge generation functions and capability learning methods.

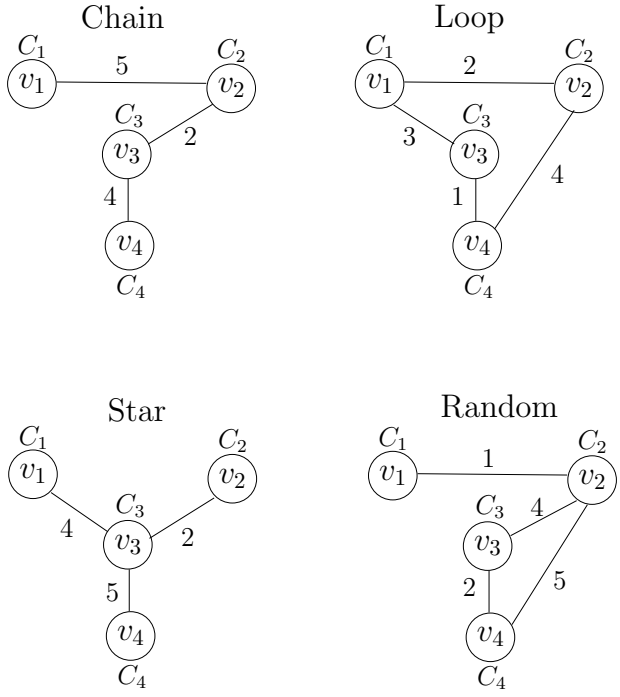


Figure 4.3: Examples of the four Weighted Synergy Graph structure types generated: chain, loop, star, and random.

agent teams that contain a_{N+1}, i.e., $\forall o = (t, v) \in O_{N+1}$, $t = (a_{N+1}, a_i)$ or $t = (a_{N+1}, a_i, a_j)$. The value v is sampled from the synergy of t (Definition 2.4.3). For each team, 30 samples are generated so 30 different observations are present per team in O_{N+1} (there are $\mathcal{O}(N^2)$ teams). We used 30 samples per team so the computed sample distributions of team performance are accurate. For the experiments using the non-linear solver, we generate 25 samples of team performances, i.e., $|O_{N+1}| = 25$. We used 25 samples for the non-linear solver as the only unknowns are the mean and variance of a_{N+1}'s capability so 25 samples should be sufficient for learning.

Algorithm 7 then adds agent a_{N+1} into the Weighted Synergy Graph to form S^+. To compare how close S^* and S^+ are, we use the KL-divergence of the synergy of possible teams: we compute the synergy (a Normally-distributed variable) of all teams containing a_{N+1}, i.e., $A \subset \mathcal{A}^+$ s.t. $|A| \geq 2$ and $a_{N+1} \in A$ and calculate the KL-divergence of the synergy using the learned S^+ from the actual S^*. The difference between S^* and S^+ is defined as the median KL-divergence of the synergies:

$$D(S^*, S^+) = \text{median}_A(D_{KL}(\mathbb{S}^*(A) \parallel \mathbb{S}^+(A))), \tag{4.1}$$

where $A \subset \mathcal{A}^+$ s.t. $|A| \geq 2$ and $a_{N+1} \in A$, and $\mathbb{S}^*$ and $\mathbb{S}^+$ use S^* and S^+ respectively.

The learning algorithm uses simulated annealing, and anneals with a fixed number of iterations, which may have an impact on the learned Weighted Synergy Graph. Hence, we ran the algorithms for both 50 and 100 iterations of simulated annealing, to determine its effects. Also, there is a random element to the learning algorithm (from the random generation of neighbor structures), and so we repeated the experiment 10 times per hidden S^*, for a total of 2000 trials (4 graph types, 50 Weighted Synergy Graphs per type, 10 trials per Weighted Synergy Graph) for each setup of the learning algorithm (capability learner and initial edge generation). We chose to perform 10 trials per hidden S^* to have a sense of the variability in the result, while keeping the total number of trials manageable.

4.5.2 Comparison Results

Table 4.1 shows the results of these experiments. We set $N = 14$, so a_{N+1} is the 15^{th} agent in the Weighted Synergy Graph that is learned. There were four Weighted Synergy Graph structure types: *Chain*; *Loop*; *Star*; and *Random*. We generated the initial edges of a_{N+1} using three heuristics: *Random* (`GenerateRandomTeammateEdges`); *Density* (`GenerateTeammateEdgesWithDensity`); and *Most Similar* (`GenerateSimilarTeammateEdges`). We used two agent capability learners, *Least-squares* and *Non-linear*, with 50 and 100 iterations of simulated annealing.

Capability Learner	Initial Edges	Weighted Synergy Graph Structure Type			
		Chain	Loop	Star	Random
Least-squares (50 iterations)	Random	16.0	47.8	3.8	7.9
	Density	14.9	33.8	4.6	7.2
	Most Similar	9.4	21.8	2.2	7.5
Least-squares (100 iterations)	Random	14.4	32.4	3.3	5.8
	Density	14.8	31.6	2.9	8.7
	Most Similar	9.3	18.6	1.7	6.8
Non-linear (50 iterations)	Random	75.3	200.0	12.7	16.6
	Density	77.3	162.0	14.2	28.2
	Most Similar	40.9	97.1	14.4	17.7
Non-linear (100 iterations)	Random	91.2	74.0	13.9	24.2
	Density	69.9	135.2	15.8	15.5
	Most Similar	45.9	72.7	14.8	17.6

Table 4.1: Average difference $D(S^*, S^+)$ between the hidden and learned Weighted Synergy Graphs given different hidden structure types and capability learning algorithms.

Comparing Agent Capability Learners

Across the capability learning functions, the average difference between the hidden Weighted Synergy Graph and the learned Weighted Synergy Graph generally improves as the number of iterations of simulated annealing goes from 50 to 100. However, when the difference is already small (e.g., with the *Star* structure type), more iterations do not improve the algorithm's performance. Between the two types of capability learners, *Least-squares* has a smaller difference than *Non-linear*, because *Least-Squares* has much more data to learn from.

Comparing Initial Edge Generation Heuristics

The three heuristics used for the initial edge generation have varying performance. The *Most similar* heuristic performs the best, learning the closest Weighted Synergy Graph across the capability learners and number of iterations. The *Random* and *Density* heuristic perform similarly, which is due to the fact that the underlying algorithm is similar except for a different density p used.

Comparing Weighted Synergy Graph Structure Types

Between the four Weighted Synergy Graph structure types, *Star* structures were the easiest to learn, followed by *Random*, while *Chain* and *Loop* structures have similar performance. The learning performed well on *Random* structures due to the random search in the simulated an-

nealing iterations, as seen from the improvement in performance between 50 and 100 iterations. The best learning performance on the *Star* structure was probably because the pairwise distances between agents do not change much with the addition or removal of new edges. In contrast, the *Chain* and *Loop* structures are more difficult to learn, since an extra edge can easily "disrupt" the structure and change the shortest distance between existing agents.

4.5.3 Comparing Different Learning Approaches

In the previous section, we analyzed the different heuristics and capability learning functions of our learning algorithm. In this section, we compare the performance of our learning algorithm against the baseline of the `LearnSynergyGraph` algorithm.

The `LearnSynergyGraph` algorithm assumes that the observation set of all the agents $a_1, \ldots, a_{N+1}$ are available initially ($O^+ = O \cup O_{N+1}$), while `AddTeammateToSynergyGraph` only requires the observations of a_{N+1} interacting with the other agents (O_{N+1}), but assumes the existence of a Synergy Graph modeling $a_1, \ldots, a_N$ and adds a_{N+1} into the Synergy Graph.

To compare these learning algorithms, we did the following: we first generate a hidden Weighted Synergy Graph S^* with $N+1$ agents, and label the vertices v_1 to v_{N+1} such that $\forall 1 \leq i \leq N+1$, the subgraph with vertices $v_1, \ldots, v_i$ remains connected, similar to the previous section. The observation sets O and O_{N+1} are then generated using S^*. We used 3 learning approaches to learn the Weighted Synergy Graph. First, in *Completely Relearn*, the `LearnSynergyGraph` algorithm is run with the complete observation set O^+. In *Learn N then Add Teammate*, we use `LearnSynergyGraph` to learn a Weighted Synergy Graph of N agents (using the observation set O, then `AddTeammateToSynergyGraph` to learn the $N+1$ agent's capability. In the third approach, *Completely Iterative*, we assume that the Weighted Synergy Graph modeling a_1 and a_2 is given (the subgraph of S^* containing two vertices), and iteratively add a_3, a_4, and so on until a_{N+1}, using `AddTeammateToSynergyGraph`.

The learned Weighted Synergy Graph from the different learning approaches are compared to the hidden one S^* using the distance function in the previous subsection (Equation 4.1). We varied the number of agents from 5 to 10, and Table 4.2 shows the results. *Completely Relearn* performs the best, as expected, with a low difference of 1.7 with 5 agents to 17.9 with 10 agents. The difference increases with the number of agents as the learning problem becomes more difficult. In comparison, *Learn N then Add Teammate* has a higher difference. However, the rate of increase in error is lower than completely relearning, which suggests that when N is large, *Learn N then Add Teammate* will perform comparably to *Completely Relearn*. Also, the runtime cost of *Completely Relearn* is much higher than that of *Learn N then Add Teammate*. The last

Synergy Graph Learner	Number of Agents					
	5	6	7	8	9	10
Completely Relearn	1.7 ± 5.1	4.4 ± 9.3	7.8 ± 12.8	13.0 ± 15.3	15.1 ± 16.3	17.9 ± 18.6
Learn N then Add Teammate	19.2 ± 52.9	18.5 ± 28.7	22.3 ± 28.1	24.0 ± 25.0	33.0 ± 63.5	29.4 ± 28.1
Completely Iterative	28.2 ± 47.9	33.3 ± 37.5	33.4 ± 31.2	43.0 ± 36.2	48.8 ± 38.9	52.3 ± 44.3

Table 4.2: Average difference between the hidden and learned Weighted Synergy Graphs using different learning methods.

approach, *Completely Iterative*, has much higher error than the other two approaches, which is due to the fact that errors accumulate as more agents are learned iteratively. Hence, it would be recommended to use *Completely Relearn* at certain intervals, so as to reset the accumulated errors, albeit at high runtime cost, and use iterative learning in small steps.

4.6 Chapter Summary

This chapter presented `AddTeammateToSynergyGraph`, an algorithm that adds a new agent teammate into an existing Synergy Graph. The algorithm assumes that observations of the new teammate and existing agents are available, and uses those observations to add the teammate into the Synergy Graph. To do so, a vertex representing the new teammate is created in the Synergy Graph, and edges are generated to connect the vertex to the other vertices in the Synergy Graph. Simulated annealing is then performed to change the edges and converge on the best edges for the new vertex. This chapter introduced three heuristics for generating the initial edges, `GenerateRandomTeammateEdges`, that randomly generates edges given a prior probability; `GenerateTeammateEdgesWithDensity`, that is similar to the previous heuristic except it estimates the edge probability using the existing Synergy Graph; and `GenerateSimilarTeammateEdges`, that finds an agent in the existing Synergy Graph that is the most similar to the new teammate, and replicates its edges. This chapter empirically compared the three heuristics, and showed that `GenerateSimilarTeammateEdges` performed the best over a variety of Weighted Synergy Graph structure types. This chapter also evaluated three different methods to learn a Synergy Graph: learning all vertices from scratch with `LearnSynergyGraph`, learning all but one vertex with `LearnSynergyGraph` then one iteration of `AddTeammateToSynergyGraph`, and learning all the vertices with `AddTeammateToSynergyGraph`. The results showed that learning the entire Synergy Graph with `LearnSynergyGraph` performs the best, but has a high runtime cost. Comparatively, learning all but one performs only slightly worse but improves the runtime, and learning all the vertices iteratively with `AddTeammateToSynergyGraph` performs poorly when there is a large number of agents.

Chapter 5

Modifications to the Synergy Graph Model

This chapter presents modifications to the Synergy Graph model that apply it to other problems such as role assignment and selecting the modules of each robot in a multi-robot team. This chapter first presents a modification to the Synergy Graph model, where each agent has multiple capabilities, and how the new Weighted Synergy Graph for Role Assignment (WeSGRA) model applies to role assignment problems [Liemhetcharat and Veloso, 2012b]. This chapter next presents modifications on the edges of the Synergy Graph, that model the interactions of robot modules within a robot and across robots in a multi-robot team, to form the Synergy Graph for Configurable Robots (SGraCR) model [Liemhetcharat and Veloso, 2013c]. This chapter also explores non-transitive relationships in the Synergy Graph model, i.e., where the shortest distance between agents is not always used in the computation of synergy, and how the non-transitive relationships affect the expressiveness of Synergy Graphs.

5.1 Agents with Multiple Capabilities

In the Synergy Graph model, each agent has an individual capability that is represented by a Normally-distributed variable, indicating the agent's contribution to the team performance. In this section, we are interested in agents that have multiple capabilities. Specifically, an agent can perform multiple *roles*, and the agent has a different capability for each role. The goal is to find a role assignment, i.e., a mapping from roles to agents, so that each role is assigned to one agent. The performance of the multi-agent team depends on which agents are assigned to which roles.

Motivating Role Assignment Scenario

As a motivating scenario, suppose that an earthquake has occurred in a city, and civilians are trapped and require rescue. Fires have started, and fallen rubble has created road blockages. An urban search-and-rescue (USAR) team is deployed in the city, with the goal of saving as many civilians as possible, and also to minimize fire damage to the city. There are three roles for the USAR agents: ambulances that can save civilians, fire engines that put out fires, and police cars that clear rubble from roads.

Many USAR agents from around the world arrive at the disaster scene ready to help, but due to safety concerns, only a small number of USAR agents can be deployed. In particular, there is a fixed set of locations within the city that the USAR agents will be deployed from. Because the USAR agents come from different places, many have not worked together before and it is unknown how well they will be able to coordinate. Thus, the goal is to pick the best ad hoc team comprising USAR agents from different sources.

5.1.1 Role Assignment Problem Definition

In the role assignment problem, we use a set of *agent types*:

Definition 5.1.1. *The **set of agent types** is* $\mathcal{A} = \{a_1, \ldots, a_N\}$.

An agent type is a combination of the hardware of a robot and the software (i.e., the algorithms) controlling it. For example, physically-identical robots running different algorithms would be represented as different agent types, while physically-identical robots running the same algorithms would be represented by a single $a \in \mathcal{A}$. Hence, N is the number of possible combinations of hardware and software, and not the total number of robots available. With software agents, each agent type $a \in \mathcal{A}$ is an algorithm, and N is the number of algorithms. When an algorithm is instantiated for a particular role, an agent is created from the agent type.

Definition 5.1.2. *The **set of roles** is* $\mathfrak{R} = \{\mathfrak{r}_1, \mathfrak{r}_2, \ldots\}$.

The task is sub-divided into multiple roles — we assume that the roles are domain-specific and given: in the USAR scenario, these roles indicate both the type of robot (ambulance, fire engine, police car) and its initial location in the city.

Using agent types and roles, we define a role assignment policy:

Definition 5.1.3. *A **role assignment policy** $\pi : \mathfrak{R} \to \mathcal{A}$ is an assignment of roles to agent types.*

For any role assignment policy π, every role must be assigned to an agent type. It is valid for the same agent type $a \in \mathcal{A}$ to be assigned multiple roles, i.e., $\exists \mathfrak{r}_\alpha, \mathfrak{r}_\beta \in \mathfrak{R}$ s.t. $\pi(\mathfrak{r}_\alpha) = \pi(\mathfrak{r}_\beta)$. Such an assignment means that multiple identical agents (e.g., robots with identical hardware and software) will each perform the role.

The agents act in a dynamic world, so the performance P_π of a role assignment policy π is non-deterministic, similar to the performance of teams in the team formation problem. Observations o_π of the performance $\mathcal{P}_\pi$ of some role assignment policies π are available. In the USAR scenario, o_π would depend on the number of civilians saved and the state of the buildings in the city.

Since this is an ad hoc scenario, the capabilities of the agent types (how well they perform at different roles) are initially unknown, in addition to how well different agent types perform together in a team. The goal is to find the δ-optimal role assignment policy, that is similar to the δ-optimal team in the team formation problem:

Definition 5.1.4. *The **δ-optimal role assignment policy** is $\pi^*_\delta : \mathfrak{R} \rightarrow \mathcal{A}$ such that there exists some utility u where π^*_δ obtains a utility of at least u with probability δ, and the probability of any other role assignment policy π doing so is at most δ:*

$$\mathbb{P}(\mathcal{P}_{\pi^*_\delta} \geq u) = \delta \textit{ and } \mathbb{P}(\mathcal{P}_\pi \geq u) \leq \delta \; \forall \pi$$

The key differences between the role assignment problem and team formation problem are:

1. A team is an assignment of roles to agent types, and not a selection of agents;
2. The size of a team is fixed, i.e., $|\mathfrak{R}|$, and every role must be assigned to an agent type.

5.1.2 Weighted Synergy Graph for Role Assignment (WeSGRA) Model

There are N agent types and $|\mathfrak{R}|$ roles, and the goal is to find the optimal role assignment policy π^*_δ. We assume that the performance of a role assignment policy is represented by a Normal distribution, similar to the performance of teams in the Synergy Graph model.

Each agent type $a_i \in \mathcal{A}$ is associated with $|\mathfrak{R}|$ Normal distributions $\left\{C_{i,1}, \ldots, C_{i,|\mathfrak{R}|}\right\}$, where each $C_{i,\alpha}$ is the individual capability of a_i at role $\mathfrak{r}_\alpha$, i.e., how well the agent type performs at the particular role. In the Synergy Graph model, agent capabilities are represented with a single Normally-distributed variable.

While the Normal distributions define the individual capabilities of agent types at different roles, it does not model how well agent types perform together. For example, suppose that in the USAR scenario, there are two roles $\mathfrak{r}_1, \mathfrak{r}_2 \in \mathfrak{R}$ involving ambulances deployed next to each other. Further suppose that agent type a_1 has high performance at $\mathfrak{r}_1$ and $\mathfrak{r}_2$, while a_2 has low performance at $\mathfrak{r}_1$ and $\mathfrak{r}_2$. However, if the role assignment policy is $(\mathfrak{r}_1 \rightarrow a_1, \mathfrak{r}_2 \rightarrow a_1)$, the overall task performance is low, because the algorithm of a_1 sends both agents to the same civilian. If the policy is $(\mathfrak{r}_1 \rightarrow a_1, \mathfrak{r}_2 \rightarrow a_2)$, the task performance is higher because different civilians would be saved. Thus, the synergy of a team is not modeled by the individual capabilities.

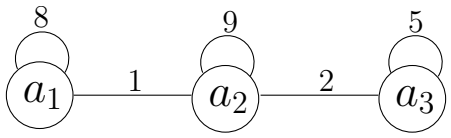


Figure 5.1: An example of the distances among three agent types $a_1, a_2, a_3 \in \mathcal{A}$, where a low distance between agent types reflects high compatibility and vice versa.

To effectively model synergy, i.e., how well a role assignment will perform, we use the compatibility function ϕ (Chapter 2), where a high compatibility reflects that agent types have good synergy and work well together. Figure 5.1 shows an example of the distances of three agent types a_1, a_2, a_3, where a_1 and a_2 have high compatibility (distance of 1) with each other, but low compatibility (distances of 8 and 9 respectively) with themselves. a_3 has high compatibility with a_2 (distance of 2). As such, through transitivity, the compatibility of a_1 and a_3 will also be moderately high (distance of 3). An agent type has compatibility with itself because multiple roles can be assigned to the same agent type (multiple agents of the same type are used, each performing a separate role), and this compatibility models how well the agents of the same type interact.

We defined the individual agent capabilities that represents how well an agent type performs at a role, and the compatibility function ϕ that relates how well pairs of agent types work together in a team. Putting these together, we now formally define the WeSGRA model:

Definition 5.1.5. *A **Weighted Synergy Graph for Role Assignment** (WeSGRA) model is a tuple (G, C), where:*

- $G = (V, E)$ *is a connected weighted graph;*
- $V = \mathcal{A}$*, i.e., the set of vertices corresponds to the set of agent types;*
- $E = E_{across} \cup E_{loop}$*, i.e., there are two types of edges, edges that go across two vertices, and self-looping edges;*
- $e_{across} = (a_i, a_j, w_{i,j}) \in E_{across}$ *is an edge between agent types a_i, a_j with weight $w_{i,j} \in \mathbb{R}^+$;*
- $\forall a_i \in \mathcal{A}, \exists e_{loop} = (a_i, a_i, w_{i,i}) \in E_{loop}$*, i.e., every agent type has a self-looping edge;*
- $\forall a_i \in \mathcal{A}, \exists C_i \in C$ *s.t.* $C_i = \left\{C_{i,1}, \ldots, C_{i,|\mathfrak{R}|}\right\}$ *where $C_{i,\alpha} \sim \mathcal{N}(\mu_{i,\alpha}, \sigma^2_{i,\alpha})$ is the capability of a_i at role $\mathfrak{r}_\alpha$.*

The WeSGRA model captures both the individual agent capabilities with C, and the compatibility ϕ between agent types through the distance between vertices in the graph.

We modify the pairwise synergy function (Definition 2.4.2) of the Synergy Graph model to apply to the WeSGRA model:

Definition 5.1.6. *The **pairwise synergy** between two agent types a_i, a_j assigned to roles $\mathfrak{r}_\alpha, \mathfrak{r}_\beta$ respectively is:*

$$\mathbb{S}_2(a_i, a_j, \mathfrak{r}_\alpha, \mathfrak{r}_\beta) = \phi(d(a_i, a_j)) \cdot (C_{i,\alpha} + C_{j,\beta}) \tag{5.1}$$

where $d(a_i, a_i) = w_{i,i}$, and for $i \neq j$, $d(a_i, a_j)$ is the shortest distance between a_i and a_j in the WeSGRA graph.

Similarly, we modify the synergy function (Definition 2.4.3) to computes the performance of a role assignment:

Definition 5.1.7. *The **synergy** of the team of agents assigned by the role policy $\pi : \mathfrak{R} \to \mathcal{A}$ is:*

$$\mathbb{S}(\pi) = \frac{1}{\binom{|\mathfrak{R}|}{2}} \cdot \sum_{\mathfrak{r}_\alpha, \mathfrak{r}_\beta \in \mathfrak{R}} \mathbb{S}_2(\pi(\mathfrak{r}_\alpha), \pi(\mathfrak{r}_\beta), \mathfrak{r}_\alpha, \mathfrak{r}_\beta) \tag{5.2}$$

Thus, the synergy function $\mathbb{S}$ returns a Normally-distributed variable that represents the performance of the team in the role assignment policy. The same agent type can have multiple roles in π, e.g., $\pi(\mathfrak{r}_\alpha) = \pi(\mathfrak{r}_\beta) = a_i$, which implies that the pairwise synergy function for this pair of roles will use the self-looping edge in the WeSGRA graph to compute the distance (and hence the compatibility).

The WeSGRA model represents heterogeneous agents in a role assignment problem, and is also applicable to homogeneous agents with different initial states. For example, suppose there is a transportation task in a city, and there are three identical agents located at different parts of the city. The role assignments decide which agent picks up which passengers for transport, and the three identical agents would be modeled as three vertices in the WeSGRA model, since the agent's different initial states create different capabilities at the roles, and hence the agents are actually heterogeneous with respect to the task.

5.1.3 Finding Role Assignments and Learning WeSGRAs

In the WeSGRA model, a team is defined as a role assignment $\pi : \mathfrak{R} \to \mathcal{A}$ that maps from a role $\mathfrak{r} \in \mathfrak{R}$ to an agent $a \in \mathcal{A}$. The δ-optimal team has an identical meaning to that in the team formation problem, i.e., the team that has the highest utility with probability δ.

The team formation algorithm `Approx`δ`OptimalTeam` is applicable to WeSGRAs with two modifications:

1. $\texttt{RandomTeam}_{\text{WesGRA}}$ replaces `RandomTeam` and returns a role assignment π such that $\pi(\mathfrak{r})$ is some random agent $a \in \mathcal{A}$.

2. `NeighborTeam`$_{\text{WeSGRA}}$ replaces `NeighborTeam` and takes an existing role assignment π and returns a neighbor role assignment π' such that:

$$\pi'(\mathfrak{r}) = \begin{cases} \pi(\mathfrak{r}) & \text{if } \mathfrak{r} \neq \mathfrak{r}' \\ a' & \text{otherwise} \end{cases} \tag{5.3}$$

for some random role $\mathfrak{r}' \in \mathfrak{R}$ and random agent $a' \in A$ such that $a' \neq \pi(\mathfrak{r}')$.

With the modified functions `RandomTeam`$_{\text{WesGRA}}$ and `NeighborTeam`$_{\text{WeSGRA}}$, the space of role assignments is explored, and the `Approx`δ`OptimalTeam` algorithm effectively approximates the δ-optimal role assignment.

The learning algorithm `LearnSynergyGraph` iterates through Synergy Graph structures and learns the agent capabilities with a least-squares solver and a non-linear solver. The algorithm is applied to the WeSGRA model with the following modifications:

1. `RandomStructure`$_{\text{WeSGRA}}$ replaces `RandomStructure` and returns a random connected weighted graph structure with additional self-looping edges of random weight;
2. `NeighborStructure`$_{\text{WeSGRA}}$ replaces `NeighborStructure`, where the random actions may also increase and decrease the weight of self-looping edges;
3. `LearnCapsLS`$_{\text{WeSGRA}}$ replaces `LearnCapabilitiesLeastSquares`, where $\mathcal{M}_\mu$ and $\mathcal{M}_{\sigma^2}$ are $\mathbf{0}_{|O|\times N|\mathfrak{R}|}$, and these matrices are filled out using Definitions 5.1.6 and 5.1.7;
4. `LearnCapsNL`$_{\text{WeSGRA}}$ replaces `LearnCapabilitiesNonLinear` and uses Definitions 5.1.6 and 5.1.7 to create the log-likelihood expression to be maximized.

5.1.4 Experiments with WeSGRAs

In the experiments that follow, we use `LearnCapabilitiesNonLinear`, that learns the agent capabilities with a non-linear solver; we ran 100 trials using synthetic data derived from a hidden WeSGRA. In each trial, we randomly created a hidden WeSGRA with 5 agent types and 5 roles, that we then used to generate 500 training and 500 test examples of $\mathcal{P}_\pi$ for different role assignment policies π. Since $N = |\mathfrak{R}| = 5$, the size of the policy space is $5^5 = 3125$ and thus the training examples do not cover the space. The training examples were used to learn a WeSGRA. The accuracy of the learned model was measured using the log-likelihood of the test examples. Next, the learned model was used to approximate the optimal role assignment policy. The value of this policy was obtained using the hidden WeSGRA, and compared to the optimal (which was found by brute force). The hidden WeSGRA was only involved in generating the training and test examples, and evaluating role assignment policies. Figure 5.2 shows the experimental process.

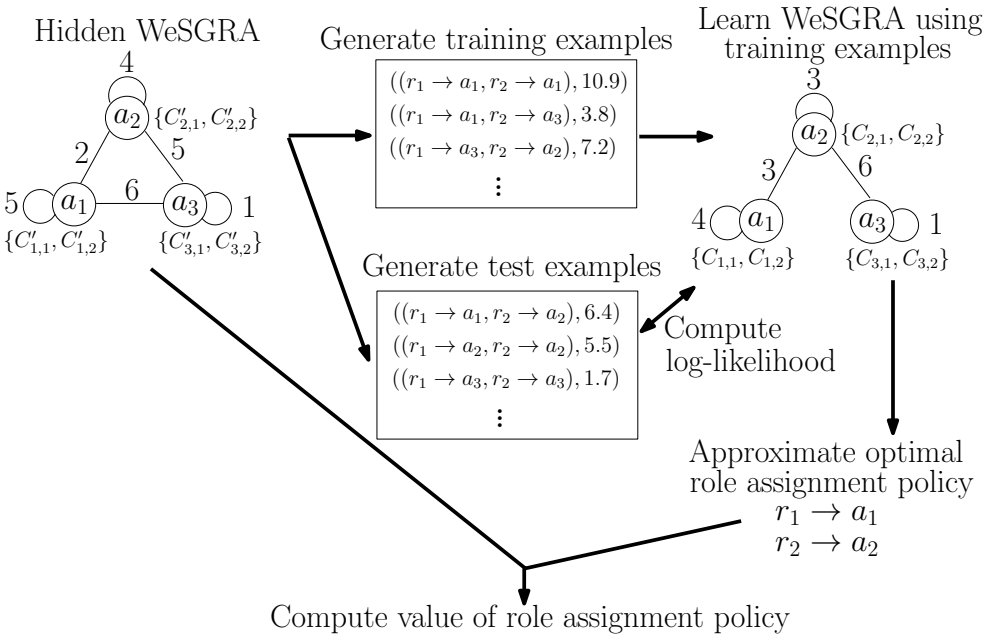


Figure 5.2: The experimental process to evaluate the learning and team formation algorithms for WeSGRA.

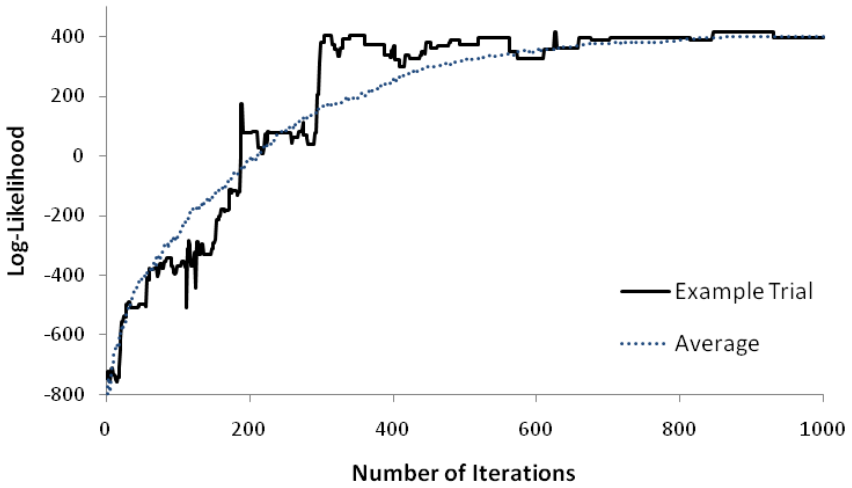


Figure 5.3: Learning curve of the learning algorithm using training examples generated by a hidden WeSGRA model.

Figure 5.3 shows the learning curve of our learning algorithm over 1000 iterations of simulated annealing. Each point on the curve shows the log-likelihood of the current learned WeSGRA on the 500 test examples. The dotted line shows the average learning over all 100 trials, while the full line shows one trial of the 100 trials. The log-likelihood of the test data becomes higher as the number of iterations increases, showing that our learning algorithm is capable of learning a WeSGRA that closely resembles the hidden one.

To compute the effectiveness of the role assignment policy selected by our team formation algorithm, for each trial, we found (through a brute-force search) the optimal and worst role assignment policies in the hidden WeSGRA, by computing the Normal distribution of every policy and converting it into a number using `Evaluate` with $\delta = \frac{1}{2}$. Next, the value of the role assignment policy found from the learned WeSGRA was also computed. Since the value of policies (optimal, minimum, and learned) differed across trials, we scaled them to be between 0 and 1:

$$\text{Effectiveness}(\pi) = \frac{\texttt{Evaluate}(\pi, \delta) - \texttt{Evaluate}(\pi_{\min}, \delta)}{\texttt{Evaluate}(\pi^*_\delta, \delta) - \texttt{Evaluate}(\pi_{\min}, \delta)} \tag{5.4}$$

where π^*_δ and $\pi_{\min}$ are the optimal and worst policies (given δ) respectively.

Over the 100 trials with synthetic data, the effectiveness of the role assignment policy found from the learned WeSGRA graph was **0.97** $\pm$ **0.08**, which shows that the learned WeSGRA closely matches the hidden one, and also that the team formation algorithm is capable of finding a near-optimal team using the learned graph.

5.2 Graphs with Multi-Edges

In the Synergy Graph model, the Synergy Graph structure is a connected graph with unweighted or weighted edges, where there is at most one edge between any two vertices. In this section, we explore graphs with multi-edges, i.e., there can be more than one edge between any two vertices. In particular, we are interested in graphs with two edges per pair of vertices, and one self-looping edge per vertex. Such multigraphs (graphs with multi-edges) are applicable to problem such as the configuration of modules of the robots in a multi-robot team, as described in this section.

Motivating Scenario for Configurable Robots

A motivating scenario comes from the manufacturing domain. High-mix low-volume manufacturing is an emerging trend, where manufacturing plants have to manufacture a large variety of products, each with a low volume. This is in stark contrast to the typical manufacturing line that produces a large volume of a single product. In order to handle high-mix low-volume or-

ders, manufacturing floors have to be reconfigurable and cater to each order. Each manufacturing station is viewed as a robot, and the entire manufacturing plant is a multi-robot system. Manufacturing stations include the typical drilling and milling, and also mobile robots that transport items. Each manufacturing robot is a configuration comprising one or more modules, e.g., a drilling robot comprises a drilling machine, and a mobile robot comprises motors and sensors. New types of robots can be configured, such as a robot that drills as it transports items.

Given a manufacturing task, the manufacturing plant has to select and configure manufacturing robots that will complete the task. The goal is to complete the task as quickly as possible, while limiting the cost of production. Robot modules have a fixed dollar cost that is borne by the manufacturing plant. Also, by assigning manufacturing modules to robots, the modules cannot be used for other orders and there is opportunity cost. Hence, the multi-robot team that is formed is capped at a maximum level of cost decided by the plant, which is a function of the dollar cost and opportunity cost. In addition, the task has a pre-defined sequence of actions (e.g., drilling followed by milling followed by polishing), and the multi-robot team has to be capable of completing the task. In particular, random combinations of modules may not be able to perform the task (e.g., selecting a robot team that does not include any drilling machines).

5.2.1 Configurable Team Formation Problem Definition

In the configurable team formation problem, robots are configured from modules, so we begin with the definition of the modules:

Definition 5.2.1. *The **set of modules** is $\mathcal{M} = M_1 \cup \ldots \cup M_N$, where each $M_n \subset \mathcal{M}$ is a set of modules of type $n \in \{1, \ldots, N\}$.*

The set of modules is divided into N types. For example, in the manufacturing scenario, M_1 would contain possible drilling modules, such that $M_1 = \{\text{drill}_0, \ldots, \text{drill}_3\}$ where the subscript refers to the number of drilling machines. M_2 would contain motors of different maximum speeds, i.e., $M_2 = \{\text{none, slow, medium, fast}\}$.

Using the modules, we define a configurable robot and the set of all robots:

Definition 5.2.2. *A **configurable robot** is $R = (m_1, \ldots, m_N)$, where each $m_n \in M_n$. The **set of all possible configurable robots** is $\mathcal{R}$.*

Thus, each robot is a configuration/selection of modules of every type. In the manufacturing example, $R_0 = (\text{drill}_2, \text{none})$ is a stationary drilling robot with two drilling machines, while a mobile transportation robot is $R_1 = (\text{drill}_0, \text{medium})$. A robot that can drill and transport items is $R_2 = (\text{drill}_1, \text{slow})$.

From the definition of robots, we define a team of robots:

Definition 5.2.3. *A **team of configurable robots** is $T = (R_1, R_2, \ldots)$, where each $R_i \in T$ is a robot configured by N modules. The **set of all possible teams** is $\mathcal{T}$.*

Each $R_i \in T$ is not unique, i.e., if $R_i = R_j$ then R_i and R_j are each configured with copies of identical modules.

Only some teams may be able to complete the task, hence we define the feasibility function:

Definition 5.2.4. *The **feasibility function** is $F : \mathcal{T} \rightarrow \{0, 1\}$, where $F(T) = 1$ if and only if the team T is feasible to complete the task, and 0 otherwise.*

The feasibility function F is domain-dependent and given as part of the problem definition.

Every module has some cost, and we define the module, robot, and team cost functions:

Definition 5.2.5. *The **module cost function** is $\mathfrak{C}_{\mathcal{M}} : \mathcal{M} \rightarrow \mathbb{R}_0^+$.*

Definition 5.2.6. *The **robot cost function** is $\mathfrak{C}_{\mathcal{R}} : \mathcal{R} \rightarrow \mathbb{R}_0^+$, such that $\mathfrak{C}_{\mathcal{R}}(R) = \sum_{i=1}^{N} \mathfrak{C}_{\mathcal{M}}(m_i)$.*

Definition 5.2.7. *The **team cost function** is $\mathfrak{C}_{\mathcal{T}} : \mathcal{T} \rightarrow \mathbb{R}_0^+$, such that $\mathfrak{C}_{\mathcal{T}}(T) = \sum_{R \in T} \mathfrak{C}_{\mathcal{R}}(R)$.*

As such, the cost of a robot is the sum of costs of its modules, and the cost of a robot team is the sum of costs of the robots in it. The module cost function is domain-dependent and part of the problem definition. In the manufacturing domain, $\mathfrak{C}_{\mathcal{M}}$ would depend on the dollar cost of the module and the opportunity cost of using the module for the task.

The performance of a robot team $T \in \mathcal{T}$ is $\mathcal{P}_T$. As we are interested in robots acting in a dynamic world, $\mathcal{P}_T$ is non-deterministic and multiple observations o_T of $\mathcal{P}_T$ return different values. In the manufacturing example, $\mathcal{P}_T$ would be related to the time required for the team T to complete the manufacturing task and the cost of the team, e.g., a task that completes earlier with a lower cost team attains a higher value; the non-determinism would be related to potential failures in the manufacturing modules.

The goal is to form the δ-optimal multi-robot team that attains the highest value subject to a maximum cost $\mathfrak{c}_{\max}$ while being feasible:

Definition 5.2.8. *The **δ-optimal team** is the team $T_\delta^* \in \mathcal{T}$ such that:*

$$F(T_\delta^*) = 1$$
$$\mathfrak{C}_{\mathcal{T}}(T_\delta^*) \leq \mathfrak{c}_{max}$$
$$\mathbb{P}(\mathcal{P}_{T_\delta^*} \geq u) = \delta$$

and for all teams $T \in \mathcal{T}$ such that $\mathfrak{C}_{\mathcal{T}}(T) \leq \mathfrak{c}_{max}$ and $F(T) = 1$,

$$\mathbb{P}(\mathcal{P}_T \geq u) \leq \delta$$

We assume that the values $\mathfrak{c}_{\max}$ and δ are given and domain-specific, as are the functions F and $\mathfrak{C}_{\mathcal{M}}$ as mentioned earlier.

5.2.2 Synergy Graph for Configurable Robots (SGraCR) Model

In the Unweighted and Weighted Synergy Graph models, agents are treated as atomic (indivisible) entities, and the agents' capabilities are modeled as Normally-distributed variables. We introduce the Synergy Graph for Configurable Robots (SGraCR) model, that is specialized to model configurable modular robots performing multi-robot tasks. We first describe each component of our model, and give the formal definition at the end of this section.

While treating agents as atomic entities allows an abstraction to capture humans and robots, we are interested in using the Synergy Graph for teams of robots. Robots are comprised of a configuration of various hardware and software modules. We model each module as a separate vertex in a graph, which offers a large benefit in scalability. From the problem definition, there are N types of modules $M_1, \ldots, M_N$, and a robot is composed of one of each type of module. In our SGraCR model, there are $\sum_{n=1}^{N} |M_n|$ vertices; the Synergy Graph model would have $\prod_{n=1}^{N} |M_n|$ vertices. For example, suppose there are 3 different types of motors, 2 different LIDARs, 3 cameras, and 2 SLAM algorithms. By modeling each module as a vertex, our SGraCR would contain $3 + 2 + 3 + 2 = 10$ vertices. A Synergy Graph models each possible type of agent/robot separately with $3 \times 2 \times 3 \times 2 = 36$ vertices. Thus, the number of vertices increases linearly in SGraCR with the number of modules, while the Synergy Graph increases geometrically.

Synergy of Modules

We are interested in modeling the task-based performance of multi-robot teams, where each robot is composed of different modules. We build upon the Synergy Graph model, where the synergy of a multi-agent team is a combination of individual agent capabilities and their compatibility in the team. Since the SGraCR models robots as composite modules, we need to differentiate between two types of synergy: intra-robot synergy and inter-robot synergy.

Intra-robot synergy models how the configuration of modules that compose a single robot affects how well the robot performs at the task. For example, a robot with faster motors performs the task quickly and attains a high performance. Comparatively, a robot with slightly slower motors but a more accurate vision system may be able to perform the task more accurately with even higher performance.

Inter-robot synergy models how different combinations of robots affect the overall task performance. Since the task requires multiple robots, different choices of robots in the team will critically affect the task performance. For example, in a foraging task, a team consisting of a single robot with accurate vision and multiple fast-moving robots that retrieve the objects may perform better than a team with multiple robots with accurate vision but move more slowly.

Similar to the Synergy Graph model, we use Normally-distributed variables to represent the capability of each module. These variables represent the contribution of task performance from the module, subject to the synergy from intra and inter-robot relationships. We use the distance between vertices in the graph to represent how well modules work together. However, there is one key difference in the SGraCR model compared to the Synergy Graph model. We distinguish between intra and inter-robot synergy by assigning two weighted edges between every pair of vertices. Conceptually, it is equivalent to an edge with two weights, an intra-robot weight and an inter-robot weight. We use the edge with two weights in our descriptions below, so that it is clearer and easier to refer to the relevant weight. Using edges with two weights implies that the underlying structure of the graph (i.e., whether edges exist between vertices) is common between intra and inter-robot synergy, even though the weights may differ. Such a representation offers a more elegant structure (a single graph structure with multiple weights) compared to two independent graphs. We believe that it is justified as there is a correlation between intra and inter-robot synergy — a module that works well for a robot will also benefit a multi-robot team.

In addition to the weighted multigraph structure, the SGraCR model has an additional edge associated with each vertex, that models the inter-robot synergy between the same module on different robots, e.g., the task performance of two robots that both have the same type of motors. An intra-robot weight in this case is not needed, since each individual robot cannot select multiple copies of the same module type.

Formal definition of the SGraCR Model

We have described the components of the SGraCR model above, and now we formally define it:

Definition 5.2.9. *The **Synergy Graph for Configurable Robots** (SGraCR) model is a tuple* (G, C)*, where:*

- $G = (V, E)$ *is a connected graph;*
- $V = \mathcal{M}$*, i.e., the set of vertices correspond to the set of modules;*
- $E = E_{multi} \cup E_{loop}$*, i.e., there are two types of edges, multi-edges and self-looping edges;*
- $e_{multi} = (m, m', w_{intra}, w_{inter}) \in E_{multi}$ *is an edge with two weights; equivalently, there are two edges between* m *and* m' *with weights* w_{intra} *and* w_{inter} *respectively.* w_{intra} *and* w_{inter} *are the intra and inter-robot weights respectively;*
- $\forall m \in \mathcal{M}, \exists e_{loop} = (m, m, w_{inter}) \in E_{loop}$*, i.e., every module has a self-looping edge with a single inter-robot weight;*
- $C = \left\{C_1, \ldots, C_{|\mathcal{M}|}\right\}$ *is a set of module capabilities, where* $C_m \sim \mathcal{N}(\mu_m, \sigma_m^2)$ *is the capability of a robotic module* $m \in \mathcal{M}$*.*

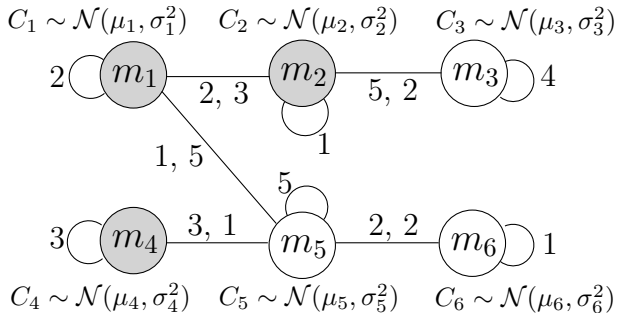


Figure 5.4: A SGraCR with 6 vertices, modeling two types of modules (shown in different shades). The edges with two weights indicate the intra and inter-robot weights respectively, and the self-looping edges have inter-robot weights.

Figure 5.4 shows an example of a SGraCR with 6 vertices, where there are two types of modules with 3 options each. Compared to a Synergy Graph that models 6 agents with 6 vertices, $3 \times 3 = 9$ different types of robots are represented with the SGraCR.

Definition 5.2.10. *The **intra-robot synergy** $\mathbb{S}_{intra}(R)$ of a robot $R = (m_1, \ldots, m_N)$ is:*

$$\mathbb{S}_{intra}(R) = \sum_{m_i, m_j \in R} \phi(d_{intra}(m_i, m_j))(C_{m_i} + C_{m_j}) \tag{5.5}$$

where C_{m_i} and C_{m_j} are the capabilities of modules m_i and m_j respectively, $d_{\text{intra}}(m_i, m_j)$ is the shortest distance between m_i and m_j in the SGraCR using the intra-weights w_{intra} of the edges, and $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ is the compatibility function.

Definition 5.2.11. *The **inter-robot synergy** $\mathbb{S}_{inter}(R, R')$ of two robots $R = (m_1, \ldots, m_N)$ and $R' = (m'_1, \ldots, m'_N)$ is:*

$$\mathbb{S}_{inter}(R, R') = \sum_{m_i \in R, m'_j \in R'} \phi(d_{inter}(m_i, m'_j))(C_{m_i} + C_{m'_j}) \tag{5.6}$$

where C_{m_i} and $C_{m'_j}$ are the capabilities of modules m_i and m'_j respectively, $d_{\text{inter}}(m_i, m'_j)$ is the shortest distance between m_i and m'_j in the SGraCR using the inter-robot weights w_{inter} of the edges. In particular, if $m_i = m'_j$ then the self-looping edge is used to determine d_{inter}.

The compatibility function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ converts distances in the SGraCR graph to real numbers reflecting the compatibility among robot modules (explained in detail in Chapter 2).

Definition 5.2.12. *The **synergy** $\mathbb{S}(T)$ of a multi-robot team T is:*

$$\mathbb{S}(T) = \frac{1}{|T|} \sum_{R \in T} \mathbb{S}_{intra}(R) + \frac{1}{\binom{|T|}{2}} \sum_{R, R' \in T} \mathbb{S}_{inter}(R, R') \tag{5.7}$$

Thus, the synergy of a multi-robot team is the sum of the average intra-robot synergy of each robot and the average inter-robot synergy of every pair of robots.

The intra and inter-robot synergy equations are modified from the pairwise synergy function of the Synergy Graph model, using the intra and inter-robot weights in the SGraCR model. The synergy function is also adapted from the Synergy Graph model, but takes into account the new definitions of intra-robot and inter-robot synergy.

5.2.3 Configuring Multi-Robot Teams and Learning SGraCRs

In the configurable team formation problem, each robot is a configuration of M modules and a multi-robot team is a set of such configurable robots. The δ-optimal team is the team that attains the highest utility with probability δ with additional constraints: the δ-optimal team T^*_δ must be feasible ($F(T^*_\delta) = 1$), within the cost threshold ($\mathfrak{C}(T^*_\delta) \leq \mathfrak{c}_{\text{max}}$). In addition, the size of the δ-optimal team is unknown, so n is no longer a parameter to `Approx`δ`OptimalTeam`. Instead, $\texttt{RandomTeam}_{\text{SGraCR}}$ iteratively increases and decreases the size of the team, and the maximum size of the team is bounded by the functions F and $\mathfrak{C}$.

$\texttt{NeighborTeam}_{\text{SGraCR}}$ generates neighbors of the existing team with three actions:

1. a random robot r_{existing} is removed,
2. a random robot r_{new} is created;
3. a module on an existing robot r_{existing} is changed.

Actions 1 and 2 (removing and creating robots) involve changing the team T. Action 1 picks a random robot $r_{\text{existing}} \in T$ and removes r_{existing} from T. Similarly, Action 2 randomly selects a robot $r_{\text{new}} \in \mathcal{R}$ and adds it to T.

Action 3 first picks a random robot $r_{\text{existing}} \in T$. Suppose $r_{\text{existing}} = (m_1, \ldots, m_M)$. Action 3 then picks a random number $i \in \{1, \ldots, M\}$ and changes module m_i in the robot to be $m'_i \neq m_i$. Hence, the new robot $r_{\text{new}} = (m_1, \ldots, m_{i-1}, m'_i, m_{i+1}, \ldots, m_M)$; r_{new} differs from r_{existing} by only one module. The robot r_{existing} is then replaced by r_{new} in T.

These 3 actions generate candidate teams that effectively explore the space of all teams, but the teams may not be feasible and/or be over the cost threshold. Thus, if $F(T) = 0$ or $\mathfrak{C}(T) > \mathfrak{c}_{\text{max}}$, the actions are repeated until a suitable team is generated. The difficulty of generating a feasible team within the cost threshold is domain-dependent since it depends on F, $\mathfrak{C}$, and $\mathfrak{c}_{\text{max}}$. Once a neighbor team is formed, its synergy is computed with $\mathbb{S}$ and converted into a real number by `Evaluate`. The new team's value is then compared to the existing and accepted subject to the temperature schedule of the simulated annealing algorithm.

`LearnSynergyGraph` is applied to the SGRaCR model with the following modifications:

1. $\texttt{RandomStructure}_{\text{SGraCR}}$ replaces `RandomStructure` and returns a random connected weighted multigraph structure with addition self-looping edges of random weight;
2. $\texttt{NeighborStructure}_{\text{SGraCR}}$ replaces `NeighborStructure`, where the random actions add and remove multi-edges, and increase and decrease the weight of the multi-edges and self-looping edges;
3. $\texttt{LearnCapsLS}_{\text{SGraCR}}$ replaces `LearnCapabilitiesLeastSquares`, where matrices $\mathcal{M}_\mu$ and $\mathcal{M}_{\sigma^2}$ are are filled out using Definitions 5.2.10, 5.2.11, and 5.2.12;
4. $\texttt{LearnCapsNL}_{\text{SGraCR}}$ replaces `LearnCapabilitiesNonLinear` and uses Definitions 5.2.10, 5.2.11, and 5.2.12 to create the log-likelihood expression to be maximized.

5.3 Non-transitive Task-Based Relationships

The Synergy Graph model assumes that the task-based relationships between agents are transitive, i.e., the compatibility of two agents depends on the shortest distance between them. Figure 5.5 shows an example of four agents in a Weighted Synergy Graph. The shortest distance between agents a_1 and a_2 is 2, using edges (a_1, a_3) and (a_2, a_3). The edge (a_1, a_2) is never used in the computation of synergy, since the transitive assumption implies that the compatibility of a_1 and a_2 should be based from the shortest distance of 2. This section explores task-based relationships that are non-transitive and how it affects the expressiveness of Synergy Graphs.

5.3.1 Modeling Non-Transitivity in Synergy Graphs

An example of a non-transitive task-based relationship is when the agents speak different languages (or use different communication protocols). Using Figure 5.5, suppose a_1 communicates via wireless LAN, a_2 communicates via bluetooth, a_4 communicates via infrared, and a_3 has all forms of communication. The edge (a_1, a_2) with a distance of 5 implies that if a_1 and a_2 were the only members of the team, they would not be able to communicate directly, and hence have a poor task-based relationship. The distances of 1 between a_1, a_2 and a_3 implies that communication is possible. Further, the distance of 2 between a_1 and a_2 (using edges (a_1, a_3) and (a_2, a_3)) implies better coordination between the two agents if a_3 is present to relay communications.

To model non-transitive task-based relationships, we consider using the shortest distance in the subgraph containing the agents in the team. The *subgraph shortest distance* approach would treat the distance between a_1 and a_2 as 5 with the team $\{a_1, a_2\}$, and 2 with the team $\{a_1, a_2, a_3\}$. An interesting related question would be the synergy of the team $\{a_1, a_4\}$ — the subgraph of a_1

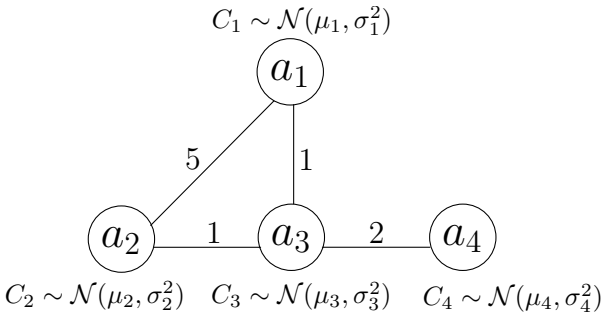


Figure 5.5: A Weighted Synergy Graph with four agents. When the task-based relationship is assumed to be transitive, the edge $\{a_1, a_2\}$ is never used in the computation of synergy.

and a_4 is disconnected, which would imply no compatibility between the two agents, i.e., they are unable to complete the task.

Hence, we define non-transitive pairwise synergy and team synergy:

Definition 5.3.1. *The **non-transitive pairwise synergy** between two agents a_i and a_j in a team $A \subseteq \mathcal{A}$ is:*

$$\mathbb{S}_{nt,2}(a_i, a_j, A) = \begin{cases} \phi(d_A(a_i, a_j)) \cdot (C_i + C_j) & \text{if } 0 < d_A(a_i, a_j) < \infty \\ -\infty & \text{otherwise} \end{cases}$$

where $d_A(a_i, a_j)$ is the shortest distance between a_i and a_j in the subgraph containing A (∞ if a_i and a_j are disconnected), and $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ is the compatibility function.

Definition 5.3.2. *The **non-transitive synergy** of a set of agents $A \subseteq \mathcal{A}$ is the average of the non-transitive pairwise synergy of its components, i.e., $\mathbb{S}_{nt}(A) = \frac{1}{\binom{|A|}{2}} \cdot \sum_{\{a_i, a_j\} \in A} \mathbb{S}_{nt,2}(a_i, a_j)$.*

The definitions of non-transitive synergy are applicable to the Unweighted and Weighted Synergy Graph models directly, and can be applied to the other Synergy Graph models with minor modifications (e.g., using role assignments instead of subsets for the WeSGRA model).

5.3.2 Implications of Non-Transitive Synergy

Non-transitive task-based relationships and the updated synergy definitions have two major implications:

1. The pairwise synergy between two agents depends on the composition of the entire team;
2. Only connected subgraphs of agents are capable of completing the task.

The first implication comes directly from the definition of non-transitive pairwise synergy; the new definition was designed to model such non-transitive task-based relationships.

The second implication is due to possible disconnected subgraphs, and having pairwise distances of ∞. The non-transitive pairwise synergy of such pairs of agents would be $-\infty$, so the non-transitive synergy of the team would also be $-\infty$.

In particular, the second implication affects the learning algorithm: teams that are in the observation set O are valid teams, so the learning algorithm has to ensure that such agent teams remain connected in their respective subgraphs. Formally, if $O_A \in O$, then the subgraph of agents in A must be connected. Hence, the functions `RandomStructure` and `NeighborStructure` of the `LearnSynergyGraph` algorithm must be updated to take into account this new constraint.

Further, while the updated learning algorithm ensures that valid teams in the observation set O form connected subgraphs, the final learned Synergy Graph structure may have connected subgraphs of invalid teams. Figure 5.6 shows an example with three agents, where the training observation set had the team $\{a_1, a_2, a_3\}$. Although a_2 and a_3 are connected in Figure 5.6a, it does not necessarily imply that the team $\{a_2, a_3\}$ is valid; Figure 5.6b shows a different Synergy Graph structure that fits the training observation set, with a_2 and a_3 disconnected.

In fact, the only valid teams that can be inferred from the observation set are those that are unions of the teams in the observations:

Theorem 5.3.3. *Let A_1 and A_2 be valid teams, i.e., the agents in A_1 and A_2 form connected subgraphs in a non-transitive Synergy Graph. If $A_1 \cap A_2 \neq \emptyset$, the team $A^+ = A_1 \cup A_2$ is also valid.*

Proof. Suppose $A_1 = \{a_{1,1}, \ldots, a_{1,n_1}\}$ and $A_2 = \{a_{2,1}, \ldots, a_{2,n_2}\}$.
Since A_1 is a valid team, $a_{1,i}$ is connected to $a_{1,j}$ $\forall 1 \leq i, j \leq n_1$ in the subgraph of A_1.
Since A_2 is a valid team, $a_{2,\alpha}$ is connected to $a_{2,\beta}$ $\forall 1 \leq \alpha, \beta \leq n_2$ in the subgraph of A_2.
Let $a \in A_1 \cap A_2$.
Since $a_{1,i}$ is connected to a in the subgraph of A_1, $a_{1,i}$ is connected to a in the subgraph of A^+.
Since $a_{2,\alpha}$ is connected to a in the subgraph of A_2, $a_{2,\alpha}$ is connected to a in the subgraph of A^+.
Therefore, $a_{1,i}$ is connected to $a_{2,\alpha}$ $\forall 1 \leq i \leq n_1, 1 \leq \alpha \leq n_2$ in the subgraph of A^+, and A^+ is a valid team. □

As a result, the team formation algorithms `Form`δ`OptimalTeam` and `Approx`δ`OptimalTeam` can only iterate on teams that are known to be valid, i.e., teams within the training observation set, and the union of teams that have common agents.

Non-transitive task-based relationships model interesting characteristics of agents, where their pairwise compatibility changes depending on the composition of the team. However, it does not model "inhibition" among the agents, e.g., an agent that, when added into a team, re-

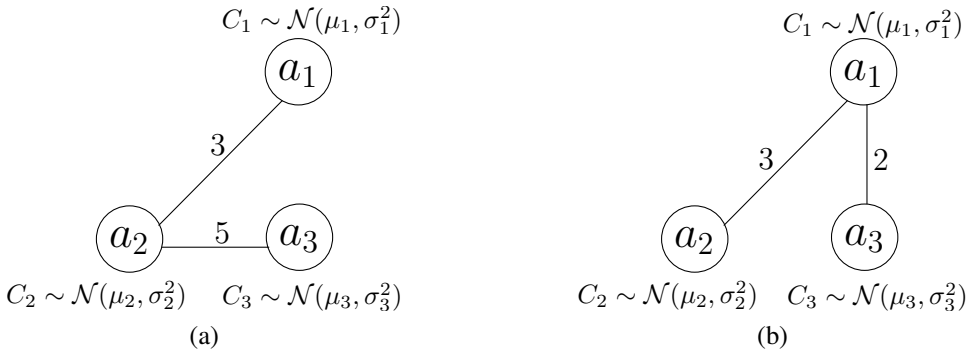


Figure 5.6: Two non-transitive Synergy Graphs where the team $\{a_1, a_2, a_3\}$ is valid.

duces the pairwise compatibility among all of other agents in the team. We are interested in cooperative, collaborative agents, and so we focus on situations where agents generally improve team performance.

5.4 Chapter Summary

This chapter presented modifications to the Synergy Graph model to apply the model to other problems such as role assignment and forming teams of configurable robots. The Weighted Synergy Graph for Role Assignment (WeSGRA) model uses agents with multiple capabilities (one for each possible role), and the Synergy Graph for Configurable Robots (SGraCR) model considers graphs with multi-edges (in particular, two weighted edges between each pair of vertices), that enable the model to capture the synergy of modules within a single robot, and the synergy of modules across robots in a multi-robot team. This chapter also presented non-transitive synergy that captures interesting characteristics, such as how a team is valid only if it forms a connected subgraph, and the implications of non-transitive synergy on the learning and team formation algorithms.

Chapter 6

Agents with Complex Characteristics

This chapter investigates agents that have complex characteristics: being susceptible to failure, and being able to improve team performance over time. First, we consider agents that probabilistically fail, i.e., there is a given probability that an agent may fail at starting the task, and define robustness and the optimal robust team [Liemhetcharat and Veloso, 2013a]. Second, we consider agents that learn to coordinate over time and improve the team performance, such as ad hoc agents [Stone et al., 2010], and explore how their improvements over time is modeled as changes in pairwise compatibility in the Synergy Graph model.

6.1 Agents that Probabilistically Fail

In this section, we consider agents that have a fixed known probability of failure. A failure occurs prior to the start of the task, and the remaining members of the team carry out the task as if the agent was not part of the team. For example, if a team consists of agents $a_1, \ldots, a_5$, and agent a_1 experiences a failure, then $a_2, \ldots, a_5$ perform the task and obtain some utility. Given such failure probabilities of agents, what is a *robust* team, and how do we form the optimal robust team? We consider an extension of the Synergy Graph for Configurable Robots (SGraCR) model where each vertex in the graph is a robot module, but the algorithms and approach applies to any Synergy Graph model.

6.1.1 Robust Team Formation Problem Definition

There is a complex task to be completed, and a robust multi-robot team is to be formed to perform the task. The goal is to form a *robust* team in the face of potential failures in the robots. To formally define this problem, we distinguish and precisely define three commonly-used terms in

multi-robot research: capability, performance, and robustness.

To aid in the explanation of the problem, we will use a consistent motivating scenario. Suppose that the task is to forage resources from the environment. There are multiple types of resources in the environment, and the multi-robot team is to find and move these resources into separate stockpiles within a limited amount of time.

We are interested in configurable robots, where every robot is configured by selecting different modules. Hence, we will use many of the definitions in Section 5.2.1. However, since we are considering possible failures in the robots, we will redefine some of the terms here to fit the robust team formation problem.

The set of modules is $\mathcal{M} = M_1 \cup \ldots \cup M_N$, where each M_n is a set of modules of a certain type (Definition 5.2.1). A configurable robot R is a configuration of N modules, i.e., $R = (m_1, \ldots, m_N)$ where each $m_n \in M_n$ (Definition 5.2.2).

Hardware and software failures can and do occur in real life, and we model the probability of success of modules:

Definition 6.1.1. *The **probability of success** $0 \leq p_m \leq 1$ of a module $m \in \mathcal{M}$ is the probability that module m does not experience a failure.*

Conversely, $1 - p_m$ is the probability that m fails and is unable to run successfully. When a module fails, a robots uses the *fall-back module*:

Definition 6.1.2. *The **fall-back module** m_n^* of type $n \in \{1, \ldots, N\}$ is a module that never fails, i.e., $p_{m_n^*} = 1 \; \forall n \in \{1, \ldots, N\}$, and is the module that a robot $R = (m_1, \ldots, m_N)$ uses if it experiences a failure of module m_n.*

For example $R = (m_1, \ldots, m_N)$ becomes $R = (m_1^*, m_2, \ldots, m_N)$ if module m_1 fails. In the foraging example, $N = 4$, where M_1 are motors, M_2 are sensors, M_3 are manipulators, and M_4 are robot bases. Some example modules in this example are: $M_1 = \{m_{\text{fast}}, m_{\text{slow}}^*\}$ and $M_4 = \{m_{\text{wheeled}}, m_{\text{treaded}}, m_{\text{legged}}, m_{\text{no base}}^*\}$. Let $R = (m_{\text{fast}}, m_{\text{camera}}, m_{\text{no manipulator}}^*, m_{\text{wheeled}})$ be an example of a wheeled robot capable of moving fast and detecting resources with its camera, but is unable to manipulate them. The module $m_{\text{no manipulator}}^*$ represents the choice of having no manipulators on the robot, and is the fall-back module of that type. Fall-back modules can be defined to indicate the absence of something (e.g., $m_{\text{no base}}, m_{\text{no manipulator}}$) or provide a baseline module (e.g., m_{slow} for motors); the modules and fall-back modules are pre-defined by the problem domain.

The probabilities p_m are independent, and multiple copies of the same module are also independent. For example, if $R_1 = (m, m')$ and $R_2 = (m, m'')$, a failure of m on R_1 is independent of a failure of m' in R_1, m in R_2, and m'' in R_2.

$\mathcal{R}$ is the set of all possible robots (Definition 5.2.2), and $T = (R_1, R_2, \ldots) \in \mathcal{T}$ is a multi-robot team (Definition 5.2.3). A robot uses a fall-back module whenever a failure occurs in one of its modules. We now define the alternatives of a robot and team:

Definition 6.1.3. *The **set of alternative robots** $\mathfrak{A}_{\mathcal{R}}(R)$ of a robot $R = (m_1, \ldots, m_N)$ is:*

$$\mathfrak{A}_{\mathcal{R}}(R) = \bigtimes_{n=1}^{N} \{m_n, m_n^*\}$$

Definition 6.1.4. *The **set of alternative teams** $\mathfrak{A}_{\mathcal{T}}(T)$ of a team $T = (R_1, R_2, \ldots)$ is:*

$$\mathfrak{A}_{\mathcal{T}}(T) = \bigtimes_{R \in T} \mathfrak{A}_{\mathcal{R}}(R)$$

The set of alternative robots contains module configurations that encompass all possible failures, e.g., if $R = (m_1, m_2)$, then $\mathfrak{A}_{\mathcal{R}}(R) = \{(m_1, m_2), (m_1^*, m_2), (m_1, m_2^*), (m_1^*, m_2^*)\}$. Similarly, the set of alternative teams contains robot configurations that encompass all possible failures. Using the set of alternative robots and teams, we define the probability of occurrence of an alternative robot and team:

Definition 6.1.5. *The **probability of occurrence of a robot** $R' \in \mathfrak{A}_{\mathcal{R}}(R)$ is:*

$$\mathbb{P}(R', R) \;=\; \prod_{m_n \in R} \begin{cases} p_{m_n} & \textit{if } m_n \in R' \\ 1 - p_{m_n} & \textit{otherwise} \end{cases}$$

Definition 6.1.6. *The **probability of occurrence of a team** $T' = (R_1', R_2', \ldots) \in \mathfrak{A}_{\mathcal{T}}(T)$ is:*

$$\mathbb{P}(T', T) \;=\; \prod_{R_n' \in T'} \mathbb{P}(R_n', R)$$

Using the set of alternative robots and teams, and the probability of occurrence, we now define the capability and performance of a team:

Definition 6.1.7. *The **capability** C_T of a team $T \in \mathcal{T}$ is the non-deterministic utility attained by T assuming that no failures occur in the modules of the robots comprising T.*

Definition 6.1.8. *The **performance** $\mathcal{P}_T$ of a team $T \in \mathcal{T}$ is the non-deterministic utility attained by T taking possible failures into account, and is a mixture model of the capabilities of the set of alternative teams of T: $\mathcal{P}_T$ has $|\mathfrak{A}_{\mathcal{T}}(T)|$ components, where each component $T' \in \mathfrak{A}_{\mathcal{T}}(T)$ has probability $\mathbb{P}(T', T)$ and distribution $C_{T'}$.*

The robots act in a dynamic world where their actions have non-deterministic outcomes. C_T captures the performance due to the non-determinism in the world, e.g., wheel slippage causing

a robot to arrive at a destination at a slower pace, and $\mathcal{P}_T$ captures both the non-determinism and the effects of failures. The capability and performance of teams are initially unknown, but observations o_T of C_T are available for some teams.

In the foraging example, C_T and $\mathcal{P}_T$ depend on the number and types of resources foraged. Suppose $|T| = 5$ and $C_T \sim \mathcal{N}(40.3, 51.7)$, i.e., the capability C_T follows a Normal distribution, and the mean utility attained is 40.3 assuming no failures occur in any of the modules of the 5 robots. Further suppose that one of modules m in one of the robots has a 0.1 probability of failure ($p_m = 0.9$), while all other modules m' are guaranteed to always work ($p_{m'} = 1$). The performance of T is then a function of C_T and $C_{T'}$, where T' is a team composed of the robots in T except that m is replaced with m^* in one of the robots. To be precise, $\mathcal{P}_T$ is a mixture model with 0.9 probability of drawing from C_T and 0.1 probability of drawing from $C_{T'}$.

The performance of a multi-robot team is a mixture model, and now we define robustness:

Definition 6.1.9. *The **robustness** $\rho(T, u)$ of a team $T \in \mathcal{T}$ is the probability that the performance $\mathcal{P}_T$ is at least a threshold utility threshold u:*

$$\rho(T, u) = \mathbb{P}(\mathcal{P}_T \geq u)$$

In the foraging example, $\rho(T, 30)$ is the probability that enough resources are foraged for at least 30 utility, considering possible module failures.

Not all multi-robot teams are capable of completing the task, e.g., as a trivial example, a robot team where all the modules have failed cannot complete any task. $F : \mathcal{T} \to \{0, 1\}$ is the feasibility function, where $F(T) = 1$ iff the team T can complete the task (Definition 5.2.4). The feasibility function F is domain-specific and we assume it is part of the problem description.

The goal is to form the optimal feasible robust team, and we introduce two measures of optimality: risk-averse optimality and risk-controlled optimality:

Definition 6.1.10. *The **risk-averse optimal team** T^*_{adv} is the team that has maximum robustness given a threshold threshold u_{thresh}:*

$$T^*_{adv} \;=\; \mathit{argmax}_{T \in \mathcal{T} \text{ s.t. } F(T)=1} \rho(T, u_{thresh})$$

Definition 6.1.11. *The **risk-controlled optimal team** T^*_{con} is the team with the highest utility with probability p_{con}:*

$$T^*_{con} \;=\; \mathit{argmax}_{T \in \mathcal{T} \text{ s.t. } F(T)=1} \{u_T | \rho(T, u_T) = p_{con}\}$$

The risk-averse optimal team and risk-controlled optimal team are two sides of the same coin: the former maximizes robustness given a utility threshold, while the latter maximizes utility given a desired robustness. The problem domain determines whether the goal is to form the risk-averse optimal team or the risk-controlled optimal team (and the associated parameters u_{thresh} and p_{con} respectively).

6.1.2 Robust Synergy Graph for Configurable Robots (ρ-SGraCR) Model

We introduced the Synergy Graph for Configurable Robots (SGraCR) model in the previous chapter, that models the capabilities of multi-robot teams composed of configurable robots. We use and extend the SGraCR model to solve the robust team formation problem by doing the following:

1. Use observations o_T to learn a SGraCR;
2. Augment the learned SGraCR with p_m to form the ρ-SGraCR model;
3. Form the optimal team using the ρ-SGraCR.

The SGraCR model effectively models the capabilities and interactions of configurable robots in a multi-robot team, but does not account for possible module failures. Hence, we augment the SGraCR model to form the Robust-SGraCR (ρ-SGraCR) model, where each vertex m in the graph is associated with the success probability p_m:

Definition 6.1.12. *The **Robust Synergy Graph for Configurable Robots** (ρ-SGraCR) model is a tuple (G, C, P), where:*

- $G = (V, E)$ *is a connected graph;*
- $V = \mathcal{M}$*, i.e., the set of vertices correspond to the set of modules;*
- $E = E_{multi} \cup E_{loop}$*, i.e., there are two types of edges, multi-edges and self-looping edges;*
- $e_{multi} = (m, m', w_{intra}, w_{inter}) \in E_{multi}$ *is an edge with two weights; equivalently, there are two edges between m and m' with weights w_{intra} and w_{inter} respectively. w_{intra} and w_{inter} are the intra and inter-robot weights respectively;*
- $\forall m \in \mathcal{M}, \exists e_{loop} = (m, m, w_{inter}) \in E_{loop}$*, i.e., every module has a self-looping edge with a single inter-robot weight;*
- $C = \{C_1, \ldots, C_{|\mathcal{M}|}\}$ *is a set of module capabilities, where $C_m \sim \mathcal{N}(\mu_m, \sigma_m^2)$ is the capability of a module $m \in \mathcal{M}$.*
- $P = \{p_1, \ldots, p_{|\mathcal{M}|}\}$ *is a set of probabilities of success of modules, where each p_m is the probability of success of module $m \in \mathcal{M}$.*

Using the ρ-SGraCR, we now solve the robust team formation problem, as described next.

6.1.3 Solving the Robust Team Formation Problem

In this section, we contribute two team formation algorithms that form a robust multi-robot team. In our description below, we assume that the goal is to find the risk-averse optimal team given a threshold u_{thresh}, i.e., to find $T^*_{\text{adv}} = \text{argmax}_{T \in \mathcal{T} \text{ s.t. } F(T)=1} \rho(T, u_{\text{thresh}})$. The algorithms would only have to be modified slightly so as to form the risk-controlled optimal team $T^*_{\text{con}} = \text{argmax}_{T \in \mathcal{T} \text{ s.t. } F(T)=1} \{u_T | \rho(T, u_T) = p_{\text{con}}\}$.

Our first team formation algorithm, `FormOptimalRobustTeam`, computes the optimal robust multi-robot team of size n^*. n^* is assumed to be known, otherwise the algorithm is run iteratively for increasing n. Algorithm 11 shows the pseudo-code of `FormOptimalRobust-Team`. The algorithm first generates all possible multi-robot teams comprising n robots. Next, the algorithm uses the synergy function $\mathbb{S}$ to compute the multi-robot team capability for all possible teams $\mathcal{T}$ and computes its score (based on the optimality function).

Algorithm 11 Find the optimal robust team with n robots

FormOptimalRobustTeam(n)

1: $\mathcal{T}_n \leftarrow$ GenerateTeams($\mathcal{M}, n$)
2: **for all** $T \in \mathcal{T}_n$ **do**
3: Score(T) $\leftarrow \mathbb{P}(\mathbb{S}(T) \geq u_{\text{thresh}})$
4: **end for**
5: **for all** $T \in \mathcal{T}_n$ **do**
6: $\rho(T, u_{\text{thresh}}) \leftarrow \sum_{T' \in \mathfrak{A}_{\mathcal{T}}(T)} \mathbb{P}(T', T)$Score($T'$)
7: **end for**
8: $T^* \leftarrow \text{argmax}_{T \in \mathcal{T}_n} \rho(T, u_{\text{thresh}})$
9: **return** T^*

Recall that $\mathfrak{A}_{\mathcal{T}}(T)$ is the set of all combinations of robots where some subset of modules work (including the superset of all modules) and some don't. To compute $\rho(T, u_{\text{thresh}})$, all combinations of module failures in the robots of T have to be considered:

$$\rho(T, u_{\text{thresh}}) = \sum_{T' \in \mathfrak{A}_{\mathcal{T}}(T)} \mathbb{P}(T', T) \mathbb{P}(\mathbb{S}(T') \geq u_{\text{thresh}}) \tag{6.1}$$

The number of teams of n robots is $|\mathcal{T}_n| = \mathcal{O}(|\mathcal{M}|^{nM})$ and $|\mathfrak{A}_{\mathcal{T}}(T)| = \mathcal{O}(2^{nM})$, where $\mathcal{M}$ is the set of all modules, and M is the number of modules in a robot, and so Algorithm 11 runs in exponential time.

Since finding the optimal robust team takes exponential time, we contribute the algorithm `ApproxOptimalRobustTeam`, that approximates the optimal robust team of size n and runs in polynomial time. Algorithm 12 shows the pseudo-code of the algorithm. Simulated annealing

is performed to limit the number of teams considered; instead of considering all possible teams of size n, only $k_{\max}$ iterations of simulated annealing are run.

Algorithm 12 Approximate the optimal robust team with n robots

ApproxOptimalRobustTeam(n)

$T^* \leftarrow$ GenerateRandomTeam($\mathcal{M}, n$)
score$^* \leftarrow$ ApproxScore(T^*)
for $k = 1$ **to** $k_{\max}$ **do**
  $T \leftarrow$ RandomNeighbor(T^*)
  score $\leftarrow$ ApproxScore(T)
  if $\mathbb{P}$(score*, score, Temp($k, k_{\max}$)) $>$ random() **then**
    $T^* \leftarrow T$
    score$^* \leftarrow v$
  end if
end for
return T^*

The function `ApproxScore`(T) approximates the score of a team T, by using the optimality function (e.g., ρ for risk-averse optimality), and only considers 3 cases: the function assumes that either none of the modules in a robot team fail, a fixed number m_{fail} of the modules fail, or all of them fail. Hence, only $\binom{nM}{m_{\text{fail}}} + 2$ combinations of teams are considered in $\mathfrak{A}'_{\mathcal{T}}(T) \subseteq \mathfrak{A}_{\mathcal{T}}(T)$ and significantly reduces the computational time to be polynomial, albeit for an approximation of the true team score:

$$\text{ApproxScore}(T) \leftarrow \frac{1}{\eta} \sum_{T' \in \mathfrak{A}'_{\mathcal{T}}(T)} \mathbb{P}(T', T)\mathbb{P}(\mathbb{S}(T') \geq u_{\text{thresh}}) \tag{6.2}$$

where $\eta = \sum_{T' \in \mathfrak{A}'_{\mathcal{T}}(T)} \mathbb{P}(T', T)$, so $\frac{1}{\eta}$ is a normalizing factor.

`GenerateRandomTeam` and `RandomNeighbor` are identical to the SGraCR team formation algorithm, so that there is effective exploration of the space of multi-robot teams. The difference in scores is compared with the temperature schedule and a random number to decide whether or not to accept a neighbor candidate.

6.1.4 Evaluating the ρ-SGraCR Model

In our first set of experiments, we use the ρ-SGraCR model to evaluate how well it solves the robust team formation problem. To do so, we generated 100 random instances of the ρ-SGraCR model, using 3 types of modules with 3 modules each (9 modules in total, $3^3 = 27$ unique configurable robots), with randomly created module capabilities and compatibility.

	No penalty		Fall-back penalty	
# of robots	Optimal Robust Team	Average team	Optimal Robust Team	Average team
1	0.577 ± 0.251	0.272 ± 0.181	0.249 ± 0.169	0.067 ± 0.052
2	0.623 ± 0.307	0.248 ± 0.231	0.134 ± 0.179	0.008 ± 0.016
3	0.653 ± 0.345	0.235 ± 0.273	0.103 ± 0.182	0.002 ± 0.006
4	0.672 ± 0.365	0.231 ± 0.300	0.089 ± 0.187	0.001 ± 0.003

Table 6.1: The optimal robustness scores of teams with 1 robot (3 modules) to 4 robots (12 modules).

We used two different settings to generate the module capabilities, *no penalty* and *fall-back penalty*. In the *no penalty* setting, all 9 module capability means were uniformly sampled from $(50, 150)$ and standard deviations were uniformly sampled from $(0, 100)$. In the *fall-back penalty* setting, the 3 fall-back modules (1 for each type) had means set to 0 and standard deviations set to 100, while other modules had means and standard deviations sampled from $(50, 150)$ and $(0, 100)$ as before. The *fall-back penalty* setting assigns the lowest capability to fall-back modules, while the *no penalty* setting has no penalties on any modules. We were interested to see if the settings would affect the optimal robust team found.

We ran the `FormOptimalRobustTeam` team formation algorithm to find the optimal robust team, from a size of 1 robot (3 modules) to 4 robots (12 modules). Table 6.1 shows the scores of the optimal teams of a fixed size. In the *no penalty* setting, increasing the number of robots in the team generally improves its robustness (the probability of at least attaining the performance threshold), while in the *fall-back penalty* setting, increasing the number of robots in the team generally decreases its robustness. The results are interesting as they show that increasing *redundancy* does not always improve *robustness*. Increasing the number of robots increases the likelihood that some of them are functional, but also increases the probability that *some* module(s) will fail and lower the team performance. As such, in the *fall-back penalty* setting, this causes the overall team score to decrease.

Thus, the ρ-SGraCR model is capable of modeling instances where the redundancy does (in the *no penalty* setting) or does not (in the *fall-back penalty* setting) improve team performance, or somewhere in between (where fall-back modules have a fraction of the other modules' capabilities), demonstrating its expressiveness in the space of robust team formation problems.

6.1.5 Comparing the Robust Team Formation Algorithms

Our second set of experiments compares the two robust team formation algorithms we contribute, `FormOptimalRobustTeam` and `ApproxOptimalRobustTeam`. The first algorithm, `FormOptimalRobustTeam`, searches for the optimal robust team of a given size n in

	No penalty		Fall-back penalty	
# of robots	Approximate Robust Team	Difference to optimal	Approximate Robust Team	Difference to optimal
1	0.565 ± 0.258	0.012 ± 0.030	0.239 ± 0.171	0.011 ± 0.021
2	0.543 ± 0.322	0.080 ± 0.103	0.109 ± 0.161	0.026 ± 0.039
3	0.556 ± 0.360	0.098 ± 0.138	0.067 ± 0.132	0.036 ± 0.077
4	0.531 ± 0.386	0.141 ± 0.190	0.042 ± 0.107	0.047 ± 0.104

Table 6.2: The robustness scores of teams formed by `ApproxOptimalRobustTeam` compared to the optimal team formed by `FormOptimalRobustTeam`.

exponential time, while `ApproxOptimalRobustTeam` approximates the optimal robust team of a given size n in polynomial time. The goal of these experiments is to empirically compare how well the approximation algorithm performs with respect to the optimal team.

Similar to the experiments above, we generated 100 random instances of ρ-SGraCR models, with 3 module types of 3 modules each (9 modules in total, 27 unique robots), in two settings, *no penalty* and *fall-back penalty*.

We used both robust team formation algorithms to compose a multi-robot team of 1 to 4 robots. There are 3654 possible 3-robot teams, and `ApproxOptimalRobustTeam` only searched 1000 teams with simulated annealing, so less than $\frac{1}{3}$ of the space was considered. Further, as an approximation, `ApproxOptimalRobustTeam` assumes that all modules do not fail, $m_{\text{fail}} = 4$ fail, or all fail, and does not consider cases in between.

Table 6.2 shows the results of our experiments. The score (the probability of at least attaining a performance threshold) of the optimal team is higher than that of the team found by `ApproxOptimalRobustTeam`, but the difference between the scores is small, which shows that `ApproxOptimalRobustTeam` performs very well considering its approximations and lower runtime.

6.2 Agents that Learn to Coordinate Better over Time

In the previous section, we considered agents that experience failures probabilistically, and how the Synergy Graph model is used to form a robust team. In this section, we investigate agents that learn to coordinate and improve team performance over time, i.e., their team performance improves as the number of learning instances they have had increases. We first formally define the problem and how the Synergy Graph model represents such learning agents. Next, we consider heuristics from the multi-armed bandit problem that apply to this learning agents problem, and empirically evaluate the performance of the heuristics.

6.2.1 Dynamic Weighted Synergy Graph (DyWeSG) Model

When agents learn to perform better over time, there are two possible reasons: the agent is learning about the *task* and improving its skill at the task; or the agent is learning to *coordinate* with its teammates better and improving the team performance. The former is represented in the Synergy Graph model as a change in the agents' capabilities, and the latter is represented by a change in the Synergy Graph structure.

We are interested in the latter, where agents learn to coordinate better with their teammates. Research on ad hoc agents (e.g., [Barrett et al., 2011]) demonstrated that changing the behavior of an agent in response to a teammate improves team performance. By considering changes in the graph structure, we are not limited to specific agents that improve; any edge in the graph (i.e., a pair of agents) can potentially improve over time. Ad hoc agents would be represented by considering that all edges connected to them can improve. Other forms of learning agents, such as specific pairs of agents that can only improve performance with each other, would be represented by only considering the edge that connects them. We consider learning agent pairs (compared to tuples of n learning agents) since the Synergy Graph uses a graph structure where edges have two agents as end-points (compared to hyper-edges that have n agents as end-points).

We use the term *learning agents* to refer to agents that learn to coordinate better with teammates, to reduce ambiguity about the term *ad hoc* as used in "ad hoc teams" in other chapters, i.e., ad hoc teams are teams where the agents have not collaborated before, so their capabilities and synergy are initially unknown.

Learning Agents Problem Definition

The set of agents is $\mathcal{A} = \{a_1, \ldots a_N\}$ (Definition 2.1.1). From the set of agents, we define learning agent pairs:

Definition 6.2.1. *A **learning agent pair** is a pair of agents $\{a_i, a_j\} \in \mathcal{A}^2$ that will improve their performance over time. The **set of learning agent pairs** is $\mathcal{L} \subseteq \mathcal{A}^2$.*

Learning agent pairs improve their performance when they are allocated training instances, which we define next:

Definition 6.2.2. *A **training instance** $k \in \{1, \ldots, K\}$ is an opportunity for a learning agent pair $\{a_i, a_j\} \in \mathcal{L}$ to improve its performance.*

Training instances are allocated to learning agent pairs, and observations are obtained:

Definition 6.2.3. *An **observation of learning** $o_{\{a_i,a_j\}}$ is obtained for each training instance that is allocated to the learning agent pair $\{a_i, a_j\} \in \mathcal{L}$.*

Since $\{a_i, a_j\}$ are learning, the value $o_{\{a_i,a_j\}}$ increases on expectation as a function of the number of training instances $\{a_i, a_j\}$ is allocated.

There are $K > 0$ training instances, and the goal is to form the optimal team of given size n^* at the end of the K instances that has the highest mean performance (i.e., the δ-optimal team with $\delta = \frac{1}{2}$). The performance of a team $A \subseteq \mathcal{A}$ of size n^* at the end of the training instances depends on the number of learning agent pairs in A, and the number of training instances each learning agent pair is allocated out of K.

Representing Coordination Improvement in Synergy Graphs

We assume that a Weighted Synergy Graph S_{initial} is given, that represents the team performance of the agents in $\mathcal{A}$ prior to the K training instances. In particular, each $\{a_i, a_j\} \in \mathcal{L}$ corresponds to an edge $(a_i, a_j, w_{i,j})$ in S_{initial}.

We formally define the Dynamic Weighted Synergy Graph (DyWeSG) model, that represents such learning edges and their improvement rate:

Definition 6.2.4. *The **Dynamic Weighted Synergy Graph** (DyWeSG) model is a tuple (G, C), where:*

- *$G = (V, E)$ is a connected weighted graph;*
- *$V = \mathcal{A}$, i.e., the set of vertices corresponds to the set of agents;*
- *$E = E_{learn} \cup E_{regular}$, i.e., there are two types of edges, learning edges and regular edges;*
- *$\forall \{a_i, a_j\} \in \mathcal{L}$, $e_{i,j} = (a_i, a_j, w_{i,j}, l_{i,j}) \in E_{learn}$ is a learning edge between agents a_i and a_j with initial weight $w_{i,j} \in \mathbb{R}^+$ and a learning rate $l_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$;*
- *$\forall \{a_i, a_j\} \notin \mathcal{L}$, $e_{i,j} = (a_i, a_j, w_{i,j}) \in E_{regular}$ is an edge between agents a_i and a_j with weight $w_{i,j} \in \mathbb{R}^+$;*
- *$C = \{C_1, \ldots, C_N\}$, where $C_i \sim \mathcal{N}(\mu_i, \sigma^2_i)$ is agent a_i's capability.*

The learning rate $l_{i,j}$ models how much the coordination between a_i and a_j improves in each training instance. We assume that the improvement rate is static and constant, and we use a Normal distribution $\mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ as our estimate of it.

Thus, from a Weighted Synergy Graph S_{initial}, a DyWeSG S is created where the agent capabilities are identical, and the graph structure is identical except for learning agent pairs. Such pairs are modeled with special edges that have an initial weight corresponding to the edge in S_{initial} and is augmented with the learning rate $l_{i,j}$.

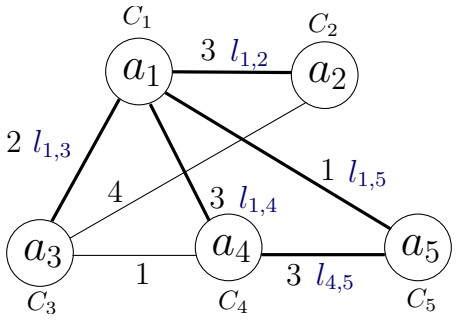


Figure 6.1: A Dynamic Weighted Synergy Graph with 5 agents. Agent pairs that learn to coordinate better over time are denoted with bold edges. Initial edge weights are shown in black text, and the learning rates are shown in blue.

Figure 6.1 shows an example DyWeSG with 5 agents, where 4 agent pairs that learn to coordinate better over time are represented with bold edges. An ad hoc agent [Stone et al., 2010] is represented with all its edges bold, such as a_1 in Figure 6.1. Other forms of learning agents, such as specific pairs of agents that only improve with respect to each other are represented only with their edges, such as $\{a_4, a_5\}$ in Figure 6.1.

Using the DyWeSG model, we compute the dynamic pairwise synergy of two agents:

Definition 6.2.5. *The **dynamic pairwise synergy** between a_i and a_j where $\{a_i, a_j\} \in \mathcal{L}$ is:*

$$\mathbb{S}_{d,2}(a_i, a_j) = (\phi(w_{i,j}) + (k_{i,j} \cdot \mu_{i,j})) \cdot (C_i + C_j)$$

where $w_{i,j}$ is the initial weight of the edge between a_i and a_j, and $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ is the compatibility function, $k_{i,j}$ is the number of training instances $\{a_i, a_j\}$ have had, and $l_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ is the estimate of the $\{a_i, a_j\}$'s learning rate.

The dynamic pairwise synergy uses the sum of the agents' capabilities, and multiplies it by the sum of the initial compatibility ($\phi(w_{i,j})$) and the learning rate multiplied by the number of training instances the agent pair has received. As such, the DyWeSG assumes that each training instance improves the compatibility of an agent pair linearly. While the compatibility improves linearly, the edge weight between them typically does not — the effective edge weight between a_i and a_j is:

$$w'_{\mathrm{i,j}} = \phi^{-1}(\phi(w_{i,j}) + (k_{i,j} \cdot \mu_{i,j}))$$

where ϕ is the compatibility function.

For any other pair of agents, the pairwise synergy (Definition 2.4.2) applies. The synergy of a team containing learning agents is:

Definition 6.2.6. *The **synergy** of a set of agents $A \subseteq \mathcal{A}$ is:*

$$\mathbb{S}_d(A) = \frac{1}{\binom{|A|}{2}} \cdot \sum_{\{a_i,a_j\} \in A} \begin{cases} \mathbb{S}_{d,2}(a_i, a_j) & \textit{if} \{a_i, a_j\} \in \mathcal{L} \\ \mathbb{S}_2(a_i, a_j) & \textit{otherwise} \end{cases}$$

Using the DyWeSG model, solving the learning agents problem has two steps:

1. Allocating K training instances, i.e., selecting K learning edges in the DyWeSG;
2. Forming the optimal team with the DyWeSG.

Comparison with the Multi-Armed Bandit Problem

There are many similarities between the learning agents problem and the multi-armed bandit problem, where each learning agent pair is viewed as an arm in the bandit problem:

1. There are a fixed number of trials/training instances;
2. Each trial improves the estimate of $l_{i,j}$;
3. There is an optimal allocation of the K trials to minimize regret.

However, there is a key difference between the two problems: the goal of the multi-armed bandit problem is to maximize the cumulative sum of rewards, while the goal of the learning agents problem is to maximize the performance of a team after the K trials.

Pulling an arm in the multi-armed bandit problem always improves the final score on expectation. In the learning agents problem, assigning an agent pair a training instance improves their compatibility, but may not affect the final team's score. For example, if the agent pair $\{a_i, a_j\}$ received $k \leq K$ training instances, but the team A that is formed after all the training instances does not contain the pair $\{a_i, a_j\}$, then the k training instances did not add to the final score.

While there is a significant difference between the multi-armed bandit problem and the learning agents problem, approaches and heuristics from the bandit problem provide insights to solving the learning agents problem, as we explain next.

6.2.2 Solving the Learning Agents Problem

The goal of the learning agents problem is to form the optimal team after K training instances. The difficulty of the problem lies in the allocation of the K training instances — after all the training instances have been allocated, the team formation algorithms `Form`δ`OptimalTeam`

(Algorithm 1) and `Approx`δ`OptimalTeam` (Algorithm 2) are applicable with $\delta = \frac{1}{2}$, with a minor change that $\mathbb{S}_d$ is used in the computation of team synergy instead of $\mathbb{S}$. In this section, we first define the optimal allocation of the K training instances, and then consider heuristics from the multi-armed bandit problem.

Computing the Optimal Allocation of Training Instances

Suppose that S^* is a DyWeSG that models the agents' team performance, where all the learning rates are known, i.e., $\forall \{a_i, a_j\} \in \mathcal{L}$, $l_{i,j} \sim \mathcal{N}(\mu_{i,j}, 0)$[1]. Since all the learning rates are known, the maximum score of any team can be computed. Algorithm 13 shows the pseudocode to find the optimal allocation of the K training instances and to form the optimal team.

Algorithm 13 Solve the learning agents problem with K training instances

SolveLearningAgentsProblem(S^*, n^*, K)

1: $A_{\text{best}} \leftarrow \emptyset$
2: $v_{\text{best}} \leftarrow -\infty$
3: $k_{\text{best}} = (k_1, \ldots, k_K) \leftarrow (\emptyset, \ldots, \emptyset)$
4: **for all** $A \subset \mathcal{A}$ **s.t.** $|A| = n^*$ **do**
5: $(v_A, k_A) \leftarrow$ `OptimalAllocation`(S^*, A, K)
6: **if** $v_A \geq v_{\text{best}}$ **then**
7: $A_{\text{best}} \leftarrow A$
8: $v_{\text{best}} \leftarrow v_A$
9: $k_{\text{best}} \leftarrow k_A$
10: **end if**
11: **end for**
12: **return** $(A_{\text{best}}, v_{\text{best}}, k_{\text{best}})$

`OptimalAllocation` computes the optimal allocation of the K training instances given a particular team A, and returns the team value and allocations k_A. To determine the optimal allocation of the K training instances, we first consider the bases cases:

1. If A contains no learning agent pairs, i.e., $\forall \{a_i, a_j\} \in \mathcal{L}$, $a_i \notin A$ or $a_j \notin A$, then k_A is irrelevant, and $v_A =$ `Evaluate`$(\mathbb{S}_d(A), \frac{1}{2})$.
2. If A contains a single learning agent pair $\{a_i, a_j\}$, then $k_A = (\{a_i, a_j\}, \ldots, \{a_i, a_j\})$, and $v_A =$ `Evaluate`$(\mathbb{S}_d(A), \frac{1}{2})$ where $k_{i,j} = K$ and $k_{\neg i,j} = k_{i,\neg j} = k_{\neg i,\neg j} = 0$.

The first case is when the team A has no learning agent pairs — no allocation of the training instances would improve A's score. The second case when A contains a single learning agent pair — all training instances are allocated to the pair to maximize the final score.

[1]Since the learning rates are known, $l_{i,j}$ is a constant value, and we use a Normal distribution with variance 0 for consistency with our previous definition of $l_{i,j}$.

In all other cases, where there are two or more learning agent pairs, the optimal allocation is to pick a single learning agent pair and allocate all K training instances to it, as shown below:

Theorem 6.2.7. *Let $\{a_i, a_j\} \in \mathcal{L}$ such that $a_i \in A$ and $a_j \in A$.*
Let $\{a_\alpha, a_\beta\} \in \mathcal{L}$ such that $a_\alpha \in A$ and $a_\beta \in A$.
Let $c_{i,j} = \mu_{i,j}(\mu_i + \mu_j)$, where $\mu_{i,j}$ is the learning rate of $\{a_i, a_j\}$, and μ_i, μ_j are the mean capabilities of the agent a_i and a_j respectively.
Let $c_{\alpha,\beta} = \mu_{\alpha,\beta}(\mu_\alpha + \mu_\beta)$, where $\mu_{\alpha,\beta}$ is the learning rate of $\{a_\alpha, a_\beta\}$, and μ_α, μ_β are the mean capabilities of the agent a_α and a_β respectively.
WLOG, suppose $c_{i,j} \geq c_{\alpha,\beta}$.
Then, $k_A = (\{a_i, a_j\}, \ldots, \{a_i, a_j\})$ is the optimal allocation of training instances.

Proof. Let $k'_A = (\{a_{1,1}, a_{1,2}\}, \ldots, \{a_{K,1}, a_{K,2}\})$ be some allocation of training instances such that $\{a_{\gamma,1}, a_{\gamma,2}\} = \{a_i, a_j\}$ or $\{a_{\gamma,1}, a_{\gamma,2}\} = \{a_\alpha, a_\beta\}$ $\forall \gamma = [1, K]$.
Let $k'_{i,j} = \sum_{\{a_{\gamma,1}, a_{\gamma,2}\} \in k'_A} \mathbb{1}_{\{a_i, a_j\}}(\{a_{\gamma,1}, a_{\gamma,2}\})$
Let $k'_{\alpha,\beta} = \sum_{\{a_{\gamma,1}, a_{\gamma,2}\} \in k'_A} \mathbb{1}_{\{a_\alpha, a_\beta\}}(\{a_{\gamma,1}, a_{\gamma,2}\})$
Let $v'_{i,j} = (\phi(w_{i,j}) + (k'_{i,j} \cdot \mu_{i,j})) \cdot (\mu_i + \mu_j)$
Let $v'_{\alpha,\beta} = (\phi(w_{\alpha,\beta}) + (k'_{\alpha,\beta} \cdot \mu_{\alpha,\beta})) \cdot (\mu_\alpha + \mu_\beta)$
$\text{argmax}_{k'_A}(\mathbb{S}_d(A)) = \text{argmax}_{k'_A}(v'_{i,j} + v'_{\alpha,\beta})$.
$v'_{i,j} + v'_{\alpha,\beta} = c + k'_{i,j} \cdot c_{i,j} + k'_{\alpha,\beta} \cdot c_{\alpha,\beta}$, where $c = \phi(w_{i,j})(\mu_i + \mu_j) + \phi(w_{\alpha,\beta})(\mu_\alpha + \mu_\beta)$.
If $c_{i,j} \geq c_{\alpha,\beta}$, then $k_A = (\{a_i, a_j\}, \ldots, \{a_i, a_j\}) = \text{argmax}_{k'_A}(v'_{i,j} + v'_{\alpha,\beta})$. □

Theorem 6.2.7 extends to agent teams with more than two learning agent pairs — it is optimal to allocate all K training instances on a single learning agent pair.

Minimizing Regret in the Learning Agent Problem

When the learning rates of the learning agent pairs are known, the optimal allocation can be computed, as shown previously. However, when the learning rates are unknown, some exploration is required to estimate the learning rates. We consider two heuristics from the multi-armed bandit problem that are applicable to the learning agents problem, the Upper Confidence Bound (UCB) heuristic [Auer et al., 2002] and the Thompson sampling (TS) heuristic [Thompson, 1933].

Algorithm 14 shows the pseudocode for the UCB heuristic on the learning agents problem. The training instances are selected by creating a modified DyWeSG where the learning agent pairs have fixed rates (Lines 6-7) and calling `SolveLearningAgentsProblem`. The learning rate of the agent pair being considered is set to the sum of the mean and standard deviation of its estimate, hence the heuristic is called upper confidence bound. `Train` applies a training

instance on the selected learning agent pair and receives an observation o_α of its performance (Line 16). The observed o_α is used to update the estimate of learning rate (Line 17).

Algorithm 14 Upper Confidence Bound heuristic on learning agents problem

UCB(S, n^*, K)

1: $k_{\text{best}} = (k_1, \ldots, k_K) \leftarrow (\emptyset, \ldots, \emptyset)$
2: **for** $\alpha = 1, \ldots K$ **do**
3: $A_{\text{best}} \leftarrow \emptyset$
4: $v_{\text{best}} \leftarrow -\infty$
5: **for all** $\{a_i, a_j\} \in \mathcal{L}$ **do**
6: $l'_{i,j} \sim \mathcal{N}(\mu_{i,j} + \sigma_{i,j}, 0)$
7: $l'_{\neg i,j} = l'_{i,\neg j} = l'_{\neg i,\neg j} \leftarrow \mathcal{N}(0, 0)$
8: $S' \leftarrow$ DyWeSG modified from S with l' as the learning rates
9: $(A_{i,j}, v_{i,j}, k_{i,j}) \leftarrow \texttt{SolveLearningAgentsProblem}(S', n^*, K - \alpha + 1)$
10: **if** $v_{i,j} \geq v_{\text{best}}$ **then**
11: $k_\alpha \leftarrow \{a_i, a_j\}$
12: $v_{\text{best}} \leftarrow v_{i,j}$
13: $A_{\text{best}} \leftarrow A_{i,j}$
14: **end if**
15: **end for**
16: $o_\alpha \leftarrow \texttt{Train}(k_\alpha)$
17: $l_{k_\alpha} \leftarrow \texttt{UpdateLearningRate}(o_\alpha)$
18: **end for**
19: **return** A_{best}

Algorithm 15 shows the pseudocode of Thompson sampling on the learning agents problem. For each learning agent pair, the best possible team of size n^* that uses the pair is computed (Lines 6–9). The performance of the team is computed, and a sample is drawn from the distribution $C_{i,j}$ (Lines 10–11). The learning agent pair with the highest sampled value is trained (Line 18), and the learning rate is updated (Line 19).

In both Algorithms 14 and 15, a crucial step is the function `UpdateLearningRate`, that updates the estimate of the learning rate of a learning agent pair. Since o_α is drawn from $\mathbb{S}_{d,2}(k_\alpha)$, which is a Normally-distributed variable, and the dynamic pairwise synergy computation is linear, a Kalman filter is used to maintain the estimate of the learning rate l_{k_α}.

6.2.3 Evaluating the Algorithms

To compare the performance of the UCB and TS heuristics on the learning agents problem, we compared the regret of both algorithms with respect to the optimal allocation of training instances. We ran 100 trials, where we generated a random Weighted Synergy Graph of 10 agents,

Algorithm 15 Thompson sampling on learning agents problem

TS(S, n^*, K)

$A_{\text{best}} = \emptyset$
$k_{\text{best}} = (k_1, \ldots, k_K) \leftarrow (\emptyset, \ldots, \emptyset)$
for $\alpha = 1, \ldots K$ **do**
  $v_{\text{best}} \leftarrow -\infty$
  for all $\{a_i, a_j\} \in \mathcal{L}$ **do**
    $l'_{i,j} \sim \mathcal{N}(\mu_{i,j}, 0)$
    $l'_{\neg i,j} = l'_{i,\neg j} = l'_{\neg i,\neg j} \leftarrow \mathcal{N}(0, 0)$
    $S' \leftarrow$ DyWeSG modified from S with l' as the learning rates
    $(A_{i,j}, \ldots) \leftarrow \texttt{SolveLearningAgentsProblem}(S', n^*, K - \alpha + 1)$
    $C_{i,j} \leftarrow \mathbb{S}_d(A_{i,j})$
    Sample $v_{i,j}$ from $C_{i,j}$
    if $v_{i,j} \geq v_{\text{best}}$ **then**
      $k_\alpha \leftarrow \{a_i, a_j\}$
      $v_{\text{best}} \leftarrow v_{i,j}$
      $A_{\text{best}} \leftarrow A_{i,j}$
    end if
  end for
  $o_\alpha \leftarrow \texttt{Train}(k_\alpha)$
  $l_{k_\alpha} \leftarrow \texttt{UpdateLearningRate}(o_\alpha)$
end for
return A_{best}

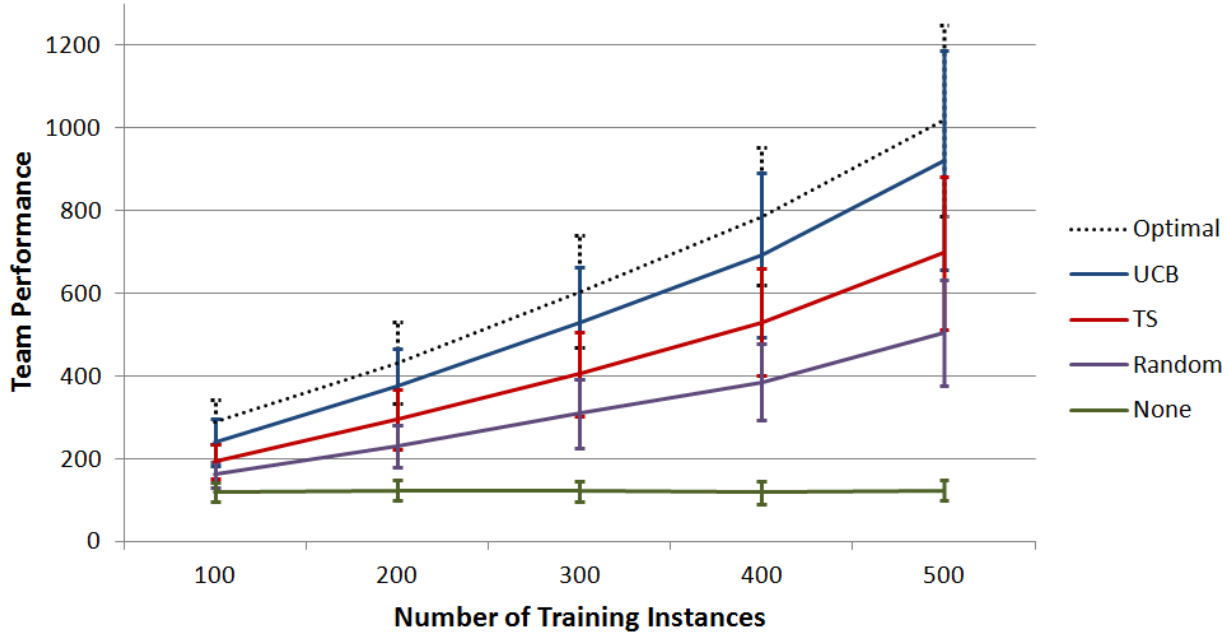


Figure 6.2: The performance of teams formed after K training instances by various heuristics. The dotted black line shows the performance of the optimal team.

where the agent capabilities $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ were generated with $\gamma = 100$, where $\mu_i \in (\frac{\gamma}{2}, \frac{3\gamma}{2})$ and $\sigma_i^2 \in (0, \gamma^2)$. We then randomly selected 5 edges to be learning agent pairs, with a hidden learning rate sampled from $(0, 0.1)$.

We varied K, the total number of training instances, from 100 to 500, and compared the UCB and TS heuristics. In addition, we ran a random heuristic that would randomly pick a learning agent pair for training, and a heuristic that did not pick any learning agent pairs at all. Figure 6.2 shows the performance of the teams formed by the various heuristics, as well as the performance of the team formed after the optimal allocation of training instances. The performance of the optimal team improves steadily as K increases, since there are more training instances for the best agent pair to improve. In contrast, the performance of the team when no learning is done remains flat. Among the other three heuristics, UCB performs the best and has a performance close to optimal. TS performs about halfway between random and UCB.

Another method of comparison is the amount of regret of each heuristic. Table 6.3 shows the regret of the heuristics compared to the optimal allocation of training instances. The regret is computed by taking the difference between the optimal team performance and the performance of the team formed by using the heuristic after the K training heuristics. Table 6.3 shows that the regret of UCB remains at around 95 after 400 training instances, while the regret of the other heuristics continue to increase. Thus, UCB outperforms the other heuristics (including TS).

Heuristic	Number of Training Instances				
	100	200	300	400	500
None	168 ± 44	308 ± 94	483 ± 134	668 ± 158	894 ± 224
Random	127 ± 32	201 ± 65	294 ± 82	401 ± 119	514 ± 172
TS	94 ± 33	137 ± 53	198 ± 71	256 ± 100	320 ± 120
UCB	48 ± 45	58 ± 46	75 ± 75	93 ± 119	97 ± 96

Table 6.3: Regret of various heuristics versus the optimal allocation of training instances.

6.3 Chapter Summary

This chapter presented how agents with complex characteristics (i.e., agents that can fail, and agents that learn to coordinate) are represented in the Synergy Graph model. We first considered agents that have probabilities of failure, and introduced the Robust Synergy Graph for Configurable Robots (ρ-SGraCR) model. We contributed two algorithms that form and approximate the optimal robust team respectively, where the optimal robust team considers the performance of the team if failures were to occur. Second, we considered agents that learn to coordinate better over time. We showed that agent pairs that perform better over time are modeled as edges in the Synergy Graph where the edge weights decrease over time, and highlighted the similarities and differences between the learning agents problem and the multi-armed bandit problem. We also contributed two algorithms that solve the learning agents problem using the upper confidence bound heuristic and Thompson sampling heuristic, and showed that the upper confidence bound heuristic forms the best-performing team, and has the lowest regret compared to the optimal allocation of training instances.

Chapter 7

Applications and Results

This chapter presents the application of the Synergy Graph model on various simulated and real robot problems. We consider team formation using a probabilistic model of robot capabilities [Parker and Tang, 2006, Liemhetcharat and Veloso, 2012a]; team formation and role assignment in RoboCup Rescue [Liemhetcharat and Veloso, 2012b, Liemhetcharat and Veloso, 2013d]; role assignment in a foraging task [Liemhetcharat and Veloso, 2012b]; configuring robot modules in a pseudo-manufacturing task [Liemhetcharat and Veloso, 2013c]; and forming a robust multi-robot team in a foraging task [Liemhetcharat and Veloso, 2013a]. This chapter presents each domain and the setup of the experiments, followed by the results of the Synergy Graph model and algorithms in forming effective teams.

7.1 Team Formation with Probabilistic Robot Capabilities

Synergy Graphs model team performance as a function of individual robot capabilities and their pairwise compatibility. Normally-distributed variables are used to capture the performance of the team as well as the robot capabilities. In this section, we use the robot capability model of Parker and Tang [Parker and Tang, 2006] for team formation, i.e., selecting a subset of the robots. We chose the probabilistic model of robot capabilities as our first benchmark as it is conceptually intuitive, and is sufficiently different from the Synergy Graph model. As such, we can evaluate the performance of the Synergy Graph model when the underlying robot capabilities and team performance does not match the Synergy Graph perfectly.

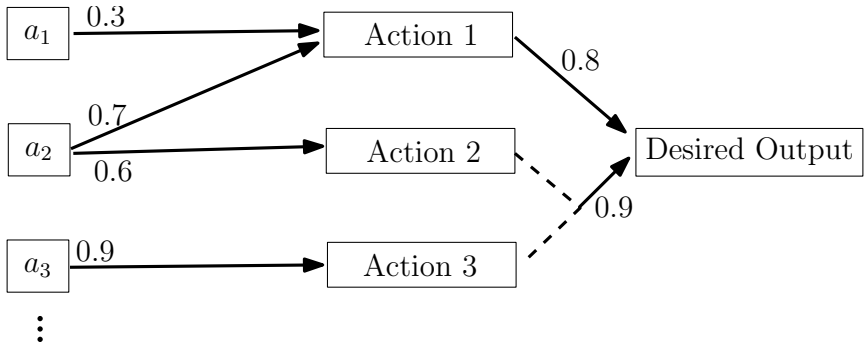


Figure 7.1: An example of probabilistic robot capabilities. The numbers indicate probabilities of success, and the dashed lines out of actions 2 and 3 indicate that both are required to trigger the desired output.

7.1.1 Probabilistic Model of Robot Capabilities

In the probabilistic model of robot capabilities, every robot has a subset of actions that it can perform, and every action by a robot has a probability of success. These actions are then chained across robots to produce the desired output, again with some probability of success. Figure 7.1 shows the capabilities of 3 agents and how the actions are chained together to produce the desired outcome. Since each agent has a subset of the actions, different subsets of agents will have different results. In our experiments, we varied the number of agents from 4 to 10 and randomly picked their capabilities in each trial — each agent had a 0.7 chance of being able to perform each action, and the probability of success of the action was uniformly sampled from $[0.1, 0.9]$.

We computed the performance of a team of agents based on the cost of executing the actions and the reward achieved by generating the output. The cost of attempting actions 1, 2 and 3 were 30, 10 and 15 respectively, and the cost of attempting to generate the output from action 1 was 10 and 15 from the combination of action 2 and 3. When the output was achieved successfully, a reward of 100 was given. The values of costs and reward were arbitrarily chosen, but further experiments with different values yielded similar results. In each trial, every agent would attempt to execute its actions, and if they were successful, the output was also attempted to be generated. Thus, the team performance had a probability density function (pdf) that depended on the agent capabilities — this pdf was not Normally distributed in general. We were interested to find out how accurately we could learn a Synergy Graph in such a situation.

7.1.2 Experimental Setup

Figure 7.2 shows the experimental process. The probabilistic model was used to generate observations of subsets of 2 and 3 agents, and then an Unweighted Synergy Graph is learned with

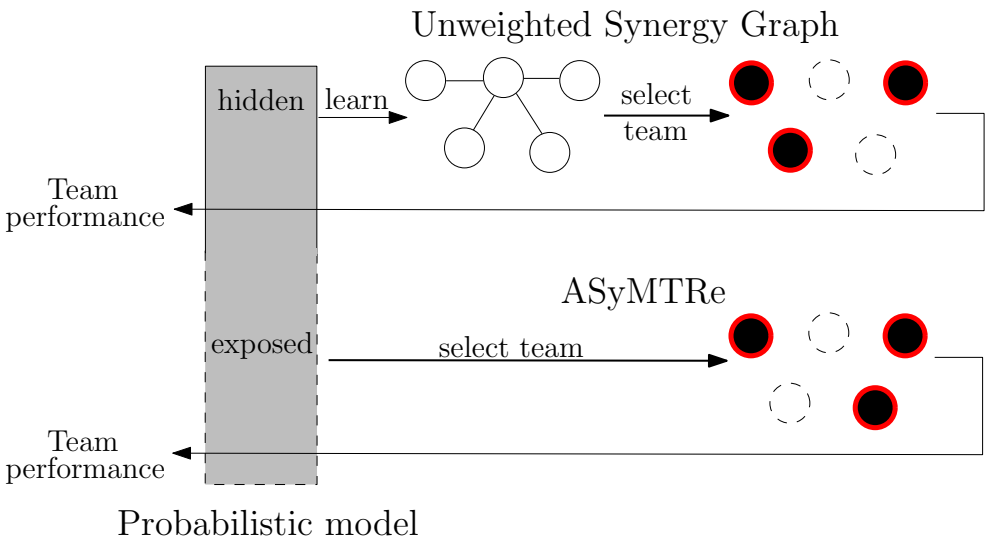


Figure 7.2: The experimental process to compare our Synergy Graph algorithms against the ASyMTRe algorithm.

`LearnSynergyGraph` (Algorithm 3). A team is selected using `Approx`δ`OptimalTeam` on the learned Unweighted Synergy Graph (Algorithm 2), and the performance of the team is computed using the probabilistic model. To attain results for ASyMTRe [Parker and Tang, 2006], the probabilities of success and costs of actions in the model were exposed, and the heuristic to rank teams in [Parker and Tang, 2006] was used. The ASyMTRe algorithm is an anytime algorithm, but for our experiments, we ran it to completion so that the optimal team with respect to the authors' heuristic was chosen. The effectiveness of the selected teams were computed by comparing their performance compared to the maximum and minimum team performance (perf_{max} and perf_{min} respectively), which were attained by performing a brute-force search of all possible combinations of agents, and thus scaling the results of the synergy and ASyMTRe algorithms to be between 0 and 1:

$$\text{Effectiveness}(A) = \frac{\text{perf}_A - \text{perf}_{\text{min}}}{\text{perf}_{\text{max}} - \text{perf}_{\text{min}}} \tag{7.1}$$

The ranking heuristic in the ASyMTRe algorithm has a factor $p \in [0, 1]$ that balances between the probability of success of performing an action versus the cost of the action. For our experiments, we varied p from 0 to 1 at 0.1 intervals, and collated the results. The Synergy Graph team formation algorithm uses $\delta \in (0, 1)$; we varied δ from 0.1 to 0.9 at 0.1 intervals and collated the results. The results were collated across p and δ since the values were consistent and had little effect on the performance of the algorithms in general. We varied $|\mathcal{A}|$, the number of agents, from 4 to 10, and picked teams of sizes 2 to $|\mathcal{A}| - 1$. For a given size of $|\mathcal{A}|$, we performed 30 trials for each team size.

# of agents	Unweighted Synergy Graph	ASyMTRe
4	0.95 ± 0.17	0.64 ± 0.34
5	0.95 ± 0.14	0.64 ± 0.33
6	0.96 ± 0.10	0.63 ± 0.31
7	0.97 ± 0.08	0.60 ± 0.29
8	0.93 ± 0.07	0.59 ± 0.27
9	0.97 ± 0.07	0.59 ± 0.25
10	0.96 ± 0.08	0.63 ± 0.27

Table 7.1: Effectiveness of teams formed by using the Unweighted Synergy Graph versus ASyMTRe.

7.1.3 Experimental Results

Table 7.1 shows the scores of the two algorithms. Across all number of agents, the Unweighted Synergy Graph outperforms ASyMTRe in terms of the effectiveness of the team selected, even though the probabilistic model of robot capabilities is hidden to the Synergy Graph algorithms but exposed for the ASyMTRe algorithm. The ASyMTRe algorithm finds teams that score around 60% of the optimal while the Synergy Graph algorithm forms teams that score above 90%. This significant difference is due to a number of reasons: firstly, the ASyMTRe algorithm was designed to also plan the agents' actions, i.e., which actions each agent should perform in order to complete the task. Secondly, the ASyMTRe typically plans for a set number of outputs (e.g., find a team to produce 2 outputs), but in our experiments the heuristic was used to find a team that produces as much output as possible. We compared our synergy algorithm to ASyMTRe as it is a well-known algorithm for multi-robot team formation and coordination that exploits heterogeneity in the agents to maximize task performance.

7.2 Team Formation in RoboCup Rescue

In this section, we use the RoboCup Rescue simulator to simulate rescue robots in a urban disaster scenario. The underlying dynamics and capabilities of the robots are hidden to the Synergy Graph algorithms, and we are interested to evaluate the performance of teams that are formed. In particular, many teams from around the world participate in the international RoboCup competition, and create task-allocation algorithms that control the rescue robots in the simulator. The experiments below treat each task-allocation algorithm as an agent, and the team formation task is to form the optimal ad hoc team, i.e., a subset of task-allocation algorithms to control the rescue robots in the RoboCup Rescue simulator.

Figure 7.3: Screenshot of the RoboCup Rescue simulator showing the initial positions of the simulated robots. Green, red, blue, and white circles are civilians, fire engines, police cars, and ambulances respectively. The grey areas indicate buildings that darken as they burn down.

7.2.1 The RoboCup Rescue Simulator

The RoboCup Rescue Simulation League provides an open-source simulator for agent development [Kitano et al., 1999]. The simulator uses a map of a city, such as Berlin, Istanbul, Paris etc, and simulates the event that a natural disaster has occurred. Civilians are randomly positioned in the city with varying amounts of health, some of whom may be buried under rubble. Fires break out in random parts of the city, and roads throughout the city are blocked by fallen debris, making them impassable for humans and vehicles.

Three types of rescue robots are deployed: ambulances, fire engines and police cars. The ambulances help to rescue civilians, while fire engines put out fires and police cars clear the road obstructions. The simulator runs for a fixed number of timesteps, and the score is a weighted sum based on the number of civilians and rescue robots alive and the proportion of buildings in the city that are not burnt.

The RoboCup Rescue domain is a multi-robot task allocation problem in the ST-MR-TA category [Gerkey and Mataric, 2004]. Different approaches have been used to solve this problem, such as treating it as a generalized allocation problem [Ferreira et al., 2010], using a biologically-inspired approach [dos Santos and Bazzan, 2011], and extending coalition formation to handle spatial and temporal constraints [Ramchurn et al., 2010].

7.2.2 Experimental Setup

As part of the RoboCup Rescue competition, participants from various universities around the world develop algorithms to control all the rescue robots (i.e., the non-civilians). We use the RoboCup Rescue simulator as an ad hoc multi-robot scenario, where combinations of pre-existing algorithms are used to compose an effective team. The rescue robots have a standardized communication protocol defined in the RoboCup Rescue simulator, which allows different RoboCup participants' algorithms to be run simultaneously, each controlling a subset of the rescue robots. In particular, we are interested in modeling the performance of ad hoc combinations of these algorithms. For example, when running two algorithms simultaneously, each algorithm controls half of the rescue robots. We wanted to compare the effectiveness of the Weighted Synergy Graph model at modeling the interactions and forming an effective team, versus the Unweighted Synergy Graph model, as well as IQ-ASyMTRe [Zhang and Parker, 2010, Zhang and Parker, 2012].

We used the Istanbul1 map from RoboCup 2011, and the source code of 6 RoboCup 2011 participants: Poseidon, RoboAKUT, Ri-one, RMAS_ArtSapience, SBCe_Saviour, and SEU_RedSun [RoboCupRescue, 2011]. The source code of 8 RoboCup participants were available for download, but only 6 ran out of the box without much modification. In the Istanbul1 map, there are 46 rescue robots to be controlled, and each of the 6 RoboCup algorithms are designed to control all 46 robots to perform the task. We treated the 6 RoboCup algorithms as 6 separate agents, such that any subset of these 6 agents can be used to control the 46 robots. We distributed the 46 rescue robots among the selected agents randomly, such that each agent controlled an approximately equal number of rescue robots. For example, if two agents (RoboCup algorithms) were picked, then each algorithm would control 23 robots (assigned randomly), and the score of the two agents would be the score returned by the RoboCup Rescue simulator at the end of its simulation, which is based on the number of civilians and rescue robots alive and the health of the buildings in the city. The RoboCup Rescue simulator models randomness and noise in the simulation, and we used the default settings of the simulator in our experiments, with the Istanbul1 map from RoboCup 2011.

We varied the number of selected agents N from 2 to 5. For each value of N, there are $\binom{6}{N}$ combinations of agents, and we ran 30 simulations for each combination. For example, when $N = 3$, there are 20 combinations of agent triples (e.g., Poseidon, RoboAKUT, and Ri-one), and we ran 30 simulations for each triple. Each simulation had a different allocation of rescue robots to agents, and hence each simulation resulted in a different score given by the simulator. An observation consists of the agents (e.g., Poseidon, RoboAKUT, and Ri-one) and a single score they attained (e.g., 14.8), and there are 30 observations per agent team combination.

We used the scores of simulation runs where $N = 2$ and $N = 3$ as the observation set for training, with $1050 = 30(\binom{6}{2} + \binom{6}{3})$ total observations. To evaluate the learned models, the algorithms formed teams of size 4 and 5. Since the score of a team changes depending on the robot allocation, we used the average score attained in the 30 simulations as our measure, hence corresponding to $\delta = \frac{1}{2}$ in our problem definition. We did not form any team of size 6, because only one such team exists (using all the agents). We did not include data from $N = 1$ for training or testing, since the simulator is deterministic (given a fixed initial random seed) so there would not be variance in the agents' performance over 30 simulations.

For the Weighted and Unweighted Synergy Graph models, we used the decay compatibility function, i.e., $\phi_{\text{decay}}(d) = \exp\left(-\frac{d \ln 2}{h}\right)$, where $h = 2$, and ran 1000 iterations of simulated annealing. We chose the decay compatibility function as the compatibility decreases more gradually than ϕ_{fraction}, and used 1000 iterations of simulated annealing as it had good results in previous experiments. In addition, since the learned Synergy Graph depends on a random process of changing edges, we performed 10 trials to learn the Synergy Graph from the RoboCup Rescue data.

IQ-ASyMTRe [Zhang and Parker, 2012] calculates the expected cost of a coalition A and task t as:

$$\text{cost}(A, t) = \widehat{\text{cost}}(A, t)/F(Q_A, Y_t) \tag{7.2}$$

where $\widehat{\text{cost}}(A, t)$ is the summation of costs of all activated schemas in A, Q_A is the coalition quality of A, Y_t is the task type of t, and $F(Q_A, Y_t)$ is the success probability of Q_A at Y_t.

Since we have a single task, and all coalitions (combinations of agents) can complete the task, we set $F(Q_A, Y_t) = 1$ for all A. As such, $\text{cost}(A, t) = \widehat{\text{cost}}(A, t)$. Since the internal schemas of the participants' algorithms are unknown, we treat each agent as a single schema, and estimate its cost as the average of the score of agent combinations involving it:

$$\widehat{\text{cost}}(a) = \frac{\sum_{A \text{ s.t. } a \in A} \text{score}(A)}{|A \in \mathcal{A} \text{ s.t. } a \in A|} \tag{7.3}$$

where a is an agent (i.e., one of the 6 algorithms), A is a team of 2 to 5 agents, and $\text{score}(A)$ is the score obtained in the RoboCup Rescue simulator using the agents in A to control the rescue robots.

From the cost of each agent, we then define the cost of a coalition in IQ-ASymTRe as:

$$\widehat{\text{cost}}(A, t) = \sum_{a \in A} \widehat{\text{cost}}(a) \tag{7.4}$$

Algorithm	4 agents	5 agents
Weighted Synergy Graph	14.3	12.7
Unweighted Synergy Graph	14.3	12.7
IQ-ASyMTRe	12.3	8.4
Best Possible Team	14.3	13.0
Worst Possible Team	7.1	8.1

Table 7.2: Average scores of combinations of algorithms in the RoboCup Rescue simulator, formed by the Weighted Synergy Graph model, Unweighted Synergy Graph model, and IQ-ASyMTRe.

To form a team using IQ-ASyMTRe, we iterate through all possible combinations of agents given the desired team size, and pick the team with the highest cost. Typically, the team with the lowest cost is picked in IQ-ASyMTRe, but because we used the score as the measure of cost (there is no actual metric for cost of the algorithms), it is desirable to pick the team with the highest score.

7.2.3 Experimental Results

Table 7.2 shows the scores of the teams formed with the Weighted Synergy Graph, Unweighted Synergy Graph, and IQ-ASyMTRe. 30 trials for each team was run in the RoboCup Rescue simulator, and the score is the average value returned by the RoboCup simulator at the end of the 30 simulations, which corresponds to using $\delta = \frac{1}{2}$.

The Weighted and Unweighted Synergy Graph models perform similarly, showing that while the Weighted Synergy Graph model is more expressive, the interactions of the agents in the RoboCup Rescue domain can be modeled with an unweighted graph. Further, both Synergy Graph models always form the optimal 4-agent team, and forms the optimal 5-agent team 80% of the time. In comparison, IQ-ASyMTRe finds a good but non-optimal 4-agent team, and forms a 5-agent team that is close to the worst possible combination. The results are statistically significant to a value of $p = 5 \times 10^{-137}$ for 4 agents, and $p = 4 \times 10^{-9}$ for 5 agents (single-tailed paired T-test) between the Synergy Graph model and IQ-ASyMTRe. The p-values for the weighted and unweighted models versus IQ-ASMTRe are identical since both Synergy Graph models attained the same results.

Thus, the Synergy Graph model outperforms IQ-ASyMTRe. The results are compelling in that only observations of 2 and 3 agents were used for training, but teams of 4 and 5 agents were formed. Further, no assumptions of the agents were used and they were treated as black-boxes in the Synergy Graph model.

7.3 Role Assignment in RoboCup Rescue

The previous section applied the Synergy Graph models to RoboCup Rescue, by treating existing RoboCup Rescue task-allocation algorithms as agents, and selecting subsets of the algorithms to control the simulated rescue robots. In this section, we treat each simulated rescue robot as a separate role, and consider the best role assignment of task-allocation algorithm to rescue robots, using the Weighted Synergy Graph for Role Assignment (WeSGRA) model.

7.3.1 Experimental Setup

The RoboCup Rescue Agent Simulation League releases the simulator, maps, and source code of participating teams annually, as described in the previous section. We compiled and ran the algorithms of 6 RoboCup teams (with minor modifications); each agent type in the WeSGRA model corresponded to a RoboCup team's algorithm. We used the Istanbul1 scenario from the actual RoboCup 2011 competition, where there are 46 rescue robots to be controlled. Typically, one team's algorithm is used to control and coordinate all 46 robots at once. However, because we are interested in modeling the interactions of multiple algorithms in an ad hoc setting, we did the following: for each of the 46 robots, one of the 6 algorithms was chosen at random to control it. Thus, a role assignment policy in this case would be an assignment of algorithms to each of the 46 rescue robot, and as such there are 6^{46} possible role assignment policies.

We randomly generated 600 policies (i.e., policies with different combinations of RoboCup algorithms), and ran an instance of the simulator per policy. At the end of each simulation, the value was retrieved from the simulator, which was a weighted sum of the health of the civilians and rescue agents still alive, and the status of the buildings of the city.

First, we performed 6-fold cross-validation on the 600 examples, where we split the data in 6 sets of 100, and trained on 500 and tested on the remaining 100. However, because of the small number of training examples, we were unable to estimate the variances of the agent type capabilities $C_{i,\alpha}$, and instead set them to be a constant. Figure 7.4 shows the cross-validation learning curves of the first fold and the average of the 6 folds, where each point on the curve is the log-likelihood of the currently learned WeSGRA (using 500 training data) on the 100 test data. The log-likelihood of the test set improves with the number of iterations of simulated annealing, and illustrates that the WeSGRA model is capable of modeling the interactions of the role assignment policies running with the 6 RoboCup algorithms.

Next, we used the learned WeSGRAs from the 6 runs of cross-validation to approximate the optimal role assignment policy. The policies found were then run in the simulator to retrieve its value. We also learned a WeSGRA using all 600 examples, and used the simulator to obtain

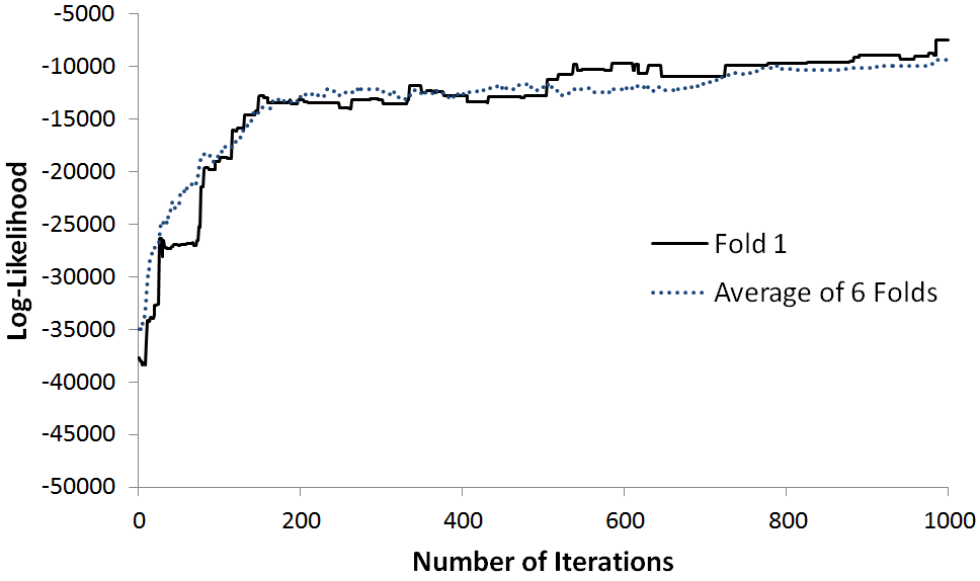


Figure 7.4: Learning curve of the Synergy Graph learning algorithm using cross-validation of data from the RoboCup Rescue simulator.

the value of the role assignment policy found from the learned WeSGRA. We compared the role assignment policies found by our approach with three methods: a random guess, a market-based approach, and picking the policy with the highest value in the training examples. The value of choosing a random policy was computed based on the 600 examples. In the market-based approach, each algorithm a_i formed a different bid $\text{Bid}(a_i, \mathfrak{r}_\alpha)$ for each of the 46 roles $\mathfrak{r}_\alpha$, based on the value of policies in the 600 examples that contained it:

$$\text{Bid}(a_i, \mathfrak{r}_\alpha) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} o_\pi \tag{7.5}$$

where $\Pi = \{\pi | \pi(\mathfrak{r}_\alpha) = a_i\}$, and o_π is the observation of the value of the role assignment π. For each role, the market-based role assignment algorithm picked the agent type with the highest bid.

7.3.2 Experimental Results

Figure 7.5 shows the distribution of values of the role assignments found by WeSGRA (from cross-validation, and from using all the data) and methods we used for comparison. The role assignments formed by WeSGRA outperforms a random policy and the market-based algorithm. The performance of WeSGRA using all the data is similar to picking the best policy in the training data, but we believe this is due to the small size of training data. WeSGRA's performance will improve with more examples, while choosing the best training policy can lead to over-fitting.

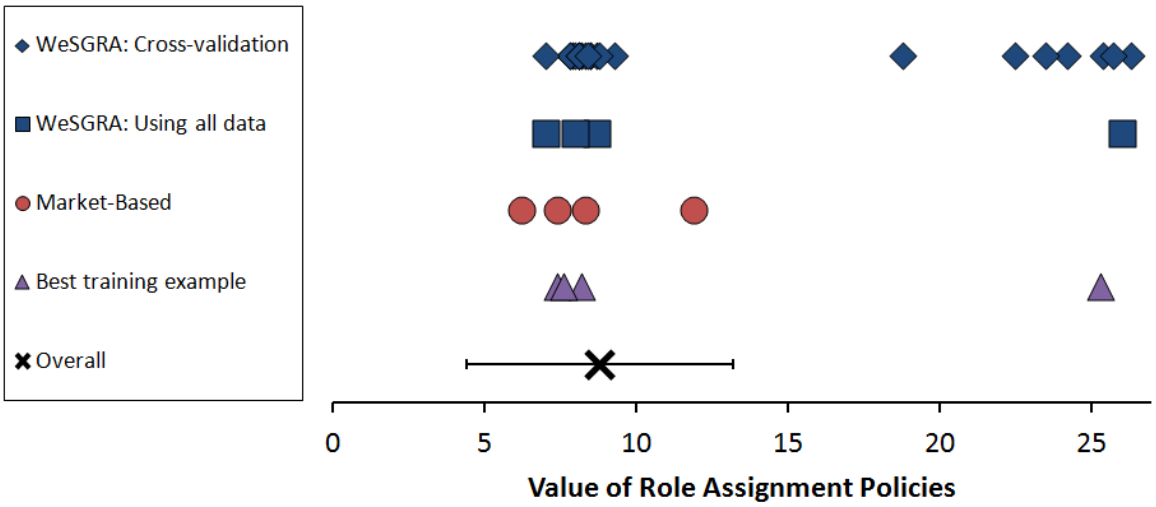


Figure 7.5: The distribution of values of role assignment policies. The values in the training examples is shown as a cross (the mean) with horizontal lines showing the standard deviation.

7.4 Role Assignment in a Foraging Task

For our third set of experiments, we applied the WeSGRA model to real robots in the foraging domain. We used two hardware platforms — Aldebaran NAO humanoid robots, and Lego Mindstorms NXT robots. While we used these two robot platforms, our results are general and can be applied to other robot types. To increase the heterogeneity in the robots, we also varied the algorithms the NAOs ran, which we will elaborate in detail below. The foraging domain was chosen as it bears many similarities to the USAR domain, namely searching and "rescuing" in a limited amount of time.

7.4.1 The Foraging Task

For the foraging task, 3 roles $\mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3$ were defined with starting locations in one half of a RoboCup Standard Platform League soccer field, where 5 balls were placed (Figure 7.6). Four of the balls were in open areas and easily seen by the NAOs, while 1 ball was hidden from view under a tunnel, thus requiring an NXT to handle it. The robots were to find and move as many balls as possible to the other half of the field, in as little time as possible. If a ball was moved outside the half of the field (i.e., the side or back lines), the ball was replaced in the middle of the half (denoted by the blue circle in Figure 7.6). The value of the team π was based on the number of balls foraged and the time in which the balls were foraged:

$$\mathrm{V}(\pi) = v_{\text{ball}} \cdot |B_\pi| + \sum_{b \in B_\pi} (t_{\text{total}} - t_b) \tag{7.6}$$

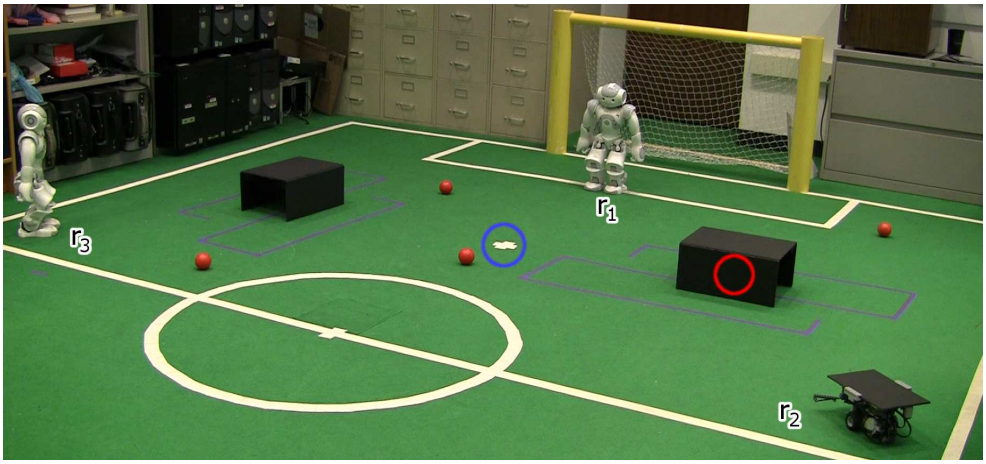


Figure 7.6: The experimental setup for the foraging experiments. The red circle indicates a hidden fifth ball, and the blue circle indicates where balls are replaced if they are moved past the side and back lines. Different combinations of robots were placed in the 3 robot roles $\mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3$.

where v_{ball} is the value per ball foraged, B are the balls foraged, t_b is the time elapsed (in seconds) when b was foraged, and t_{total} is the total time of the trial in seconds.

7.4.2 Experimental Setup

The NAO robots had three different algorithms: Chase-and-Kick (CK), Kick-past-Midline (KM), and Observer (Ob). A NAO running CK would search for a ball, walk to it, and then perform a straight kick in whichever direction the robot was currently facing. In the KM algorithm, the NAO would explicitly localize (based on landmarks on the field such as the yellow and blue goal posts) and attempt to kick the ball past the middle line. Thus, the KM algorithm explicitly kicks the ball towards the target area (the other half of the field), while the CK algorithm kicks in any direction, but over time CK succeeds in foraging since balls kicked past the side and back lines are replaced. Also, in both algorithms, the NAO would perform obstacle avoidance so as not to walk into another NAO, NXT or the tunnels.

The Ob algorithm of the NAO did not actively forage the balls — instead, the NAO would search for balls, and transmit the ball's position through a wireless connection to other NAOs in the team, and as such potentially reducing the amount of time another NAO needed to find a ball.

The NXT robots were programmed to follow lines, such that they moved straight across the green field, turned around when they encountered a white line, and followed a purple line. Thus, the NXT robot would move across the field until it found one of the two sets of purple lines around a tunnel, and then followed it endlessly (it turned around at the end of the purple line). If a ball was present inside the tunnel, the NXT would push it out as part of the line-following

behavior. As such, the NXT robots were not capable of foraging the balls directly, but only assisted the overall team goal.

The robots in our experiments ran autonomously, without any central computer or processing. The only information communicated between the robots was the ball position that the Ob algorithm sent to KM, that KM would use to approach the ball. We chose these algorithms so that there was a greater heterogeneity in the robots, and also that some algorithms were explicitly "helper" types (i.e., Ob and NXTs), where they would not attain any value on their own, but can improve the overall value given the right teammates.

There were 3 roles, and 4 possible agent types (CK, KM, Ob, NXT), and thus there were $4^3 = 64$ possible role assignment policies. We performed 61 of these policies and recorded the times in which balls were successfully foraged (3 combinations involving all NAOs could not be run due to hardware problems).

Next, we set $v_{\text{ball}} = 100$, and computed the values of each of the combinations, given different amounts of time per trial. For example, if t_{total} was 120 seconds, then we ignored all balls foraged after 120 seconds. We varied t_{total} from 120 to 600 seconds at 60 second intervals.

For each value of t_{total}, we performed 6-fold cross-validation on the data, comparing the WeSGRA model (using 4 agent types and 3 roles) with the market-based algorithm (Equation 7.5). Due to the small number of training examples, we again fixed the variances in the WeSGRA model to a constant. As t_{total} increases, the values of policies found by both algorithms increase, as there is more time for the robots to forage balls. To compare the performance of the algorithms, we used the effectiveness measure to scale the results from 0 to 1:

$$\text{Effectiveness}(\pi) = \frac{V(\pi) - V(\pi_{\text{min}})}{V(\pi_{\text{max}}) - V(\pi_{\text{min}})} \tag{7.7}$$

where π_{max} and π_{min} are the optimal and worst role assignment policies respectively.

7.4.3 Experimental Results

Table 7.3 shows the results of WeSGRA and the market-based algorithm. WeSGRA outperforms the market-based algorithm across all values of t_{total}. Using a single-tailed paired-sample T-test, we found that our results are statistically significant with $p = 3 \times 10^{-5}$.

The market-based technique picked the role assignment ($\mathfrak{r}_1 \rightarrow$ CK, $\mathfrak{r}_2 \rightarrow$ KM, $\mathfrak{r}_3 \rightarrow$ KM) regardless of t_{total} and as such the team could only forage the 4 visible balls. The WeSGRA model picked teams involving 1 NXT and 2 NAOs (running CK and KM), and were thus able to forage more balls in general and attain a higher value. Thus, the WeSGRA model successfully modeled the interactions between the helper types of robots. While the Ob algorithm helped to

Algorithm	t_{total}								
	120	180	240	300	360	420	480	540	600
WeSGRA	0.91 ± 0.11	0.92 ± 0.11	0.96 ± 0.04	0.93 ± 0.08	0.92 ± 0.07	0.93 ± 0.06	0.88 ± 0.06	0.88 ± 0.06	0.90 ± 0.08
Market-based	0.80 ± 0	0.80 ± 0	0.88 ± 0	0.93 ± 0	0.90 ± 0	0.89 ± 0	0.87 ± 0	0.86 ± 0	0.86 ± 0

Table 7.3: Effectiveness of algorithms in the foraging domain with real robots.

provide ball information to the other NAOs, it was not selected as part of the optimal team, as having more NAOs walking in the field provides a larger benefit.

7.5 Configuring a Team for a Manufacturing Task

In this section, we apply the Synergy Graph for Configurable Robots (SGraCR) model to a pseudo-manufacturing scenario. We define various modules that a configurable robot uses for manufacturing and transportation, with the goal of maximizing the team performance and keeping the total cost of the modules below a threshold.

7.5.1 Experimental Setup

In all three sets of experiments below, we used the same problem domain — manufacturing. The task involved transporting some items from location L_0 to perform drilling (at location L_1) and then milling (at location L_2) and finally delivered to a destination location L_3. The manufacturing floor had pre-existing drilling and milling stations at fixed locations (L_1 and L_2 respectively), and the goal was to form a multi-robot team that would move all the items through the manufacturing plan. All the robots would be mobile, and had varying behaviors. Robots could also be configured to perform drilling *or* milling, which would allow the items to be transported past locations. For example, if a robot could drill, then it could transport items from L_0 directly to L_2 for milling, bypassing L_1 since the robot performs the drilling.

We defined 3 types of modules: M_1, M_2, M_3. $M_1 = \{\text{slow, medium, fast}\}$ are the motors, $M_2 = \{1, 2, 3, 4\}$ are the carrying capacities, and $M_3 = \{B_{0,1}, B_{0,2}, B_{1,2}, B_{1,3}, B_{2,3}\}$ encapsulate both the software programmed into the robots and drilling/milling capabilities. A behavior $B_{i,i+1}$ implies that the robot only transports items from L_i to L_{i+1}. A behavior $B_{i,i+2}$ implies that the robot also performs drilling/milling, e.g., $B_{1,3}$ means that a robot transports drilled items from L_1 to location L_3 and performs milling on the item.

The feasibility function returns 1 iff the team of robots are able to drill, mill and transport all items to L_3. For all our experiments below, we set $\delta = \frac{1}{2}$, so the goal was to find the team that attains the highest mean value. Faster motor speeds, higher carrying capacities, and adding drilling/milling functionality had higher costs. The cost threshold $\mathfrak{c}_{\text{max}}$ was set such that the

Approach	Score		
	Synthetic Data	Simulation	Real Robots
SGraCR	1.77 ± 1.64	1.33 ± 0.52	0.86 ± 0.46
Unweighted Synergy Graph	0.56 ± 1.45	1.33 ± 0.29	0.37 ± 0.82
IQ-ASyMTRe	0.93 ± 1.99	0.41 ± 0.54	0.59 ± 0.23

Table 7.4: Experimental results of SGraCR and two competing approaches using synthetic data derived from a hidden SGraCR model, simulated robots in a manufacturing scenario, and robot experiments using Lego NXT robots. The scores indicate the number of standard deviations above the mean, i.e., a score of x means that the approach found a team with a value $\mu + x\sigma$, where μ and σ are the mean and standard deviation of values of teams.

maximum number of robots was five for the synthetic and simulation experiments, and three for the real robot experiments.

In each trial, we generated a set of training data. The learning algorithm uses the training data to learn a SGraCR model, and the team formation algorithm uses the learned SGraCR to find the team that approximates the δ-optimal team. A similar approach was used to learn a Synergy Graph and form a team, and for IQ-ASyMTRe the training data set was used to estimate the module costs.

7.5.2 Experiments with Synthetic Data

In our first set of experiments, we used synthetic data derived from a hidden SGraCR model. Using the experimental domain described above, we generated a hidden SGraCR model with 12 vertices and randomly generated the module capabilities. The hidden model was used to create 100 training data $(T, V(T)) \in O_{\text{train}}$, where the value $V(T) = \texttt{Evaluate}(\mathbb{S}(T), 1 - \delta)$ of the hidden model. The training data was used to learn a new SGraCR model. Finally, our team formation algorithm was run on the learned SGraCR model, and its value calculated using the hidden model, i.e., if the algorithm selected team T, then $V(T) = \texttt{Evaluate}(\mathbb{S}(T), 1 - \delta)$ of the *hidden* model.

We performed 20 trials, where a different hidden SGraCR model was generated each time. The *Synthetic Data* column of Table 7.4 shows the results of our trials with synthetic data. SGraCR outperformed the Unweighted Synergy Graph model and IQ-ASyMTRe. We believe that this is largely because the data was derived from a hidden SGraCR model. The low performance of the Unweighted Synergy Graph compared to SGraCR shows that SGraCR is a more expressive model; otherwise, the Synergy Graph would have a similar score to SCraCR.

7.5.3 Experiments with Simulated Robots

In our second set of experiments, we created a 2D simulator, where mobile robots moved to transport items from one location to another. The value of a team was the negative of the number of timesteps taken to transport 100 items from L_0 to L_3, i.e., if a team T took t timesteps then $V(T) = -t$.

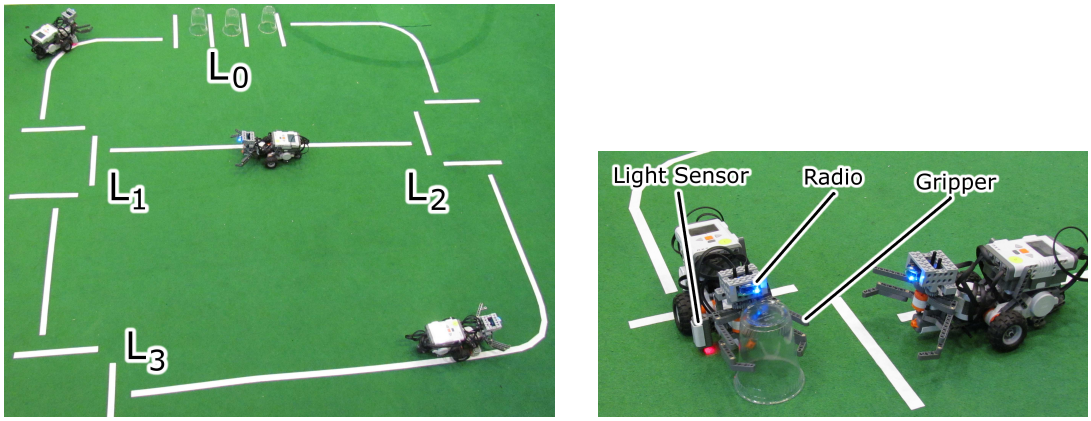
We ran the simulator on all 6056 possible teams to calculate their value. These 6056 values were scaled to form the complete data set for the experiment, where a score of x means the team had a value of $\mu + x\sigma$, where μ and σ are the mean and standard deviation of the values of all 6056 teams. We ran 20 trials where in each trial, 100 data points of the 6056 was used for training, so only a small subset of possible teams was visible by the learning algorithm to learn a SGraCR. The team formation algorithm then searched the learned SGraCR to approximate the δ-optimal team. The score of the formed team was then retrieved from the 6056 data points.

The *Simulation* column of Table 7.4 shows the results of the simulated experiments. Both the SGraCR and Unweighted Synergy Graph models perform very well, finding teams with scores of 1.33, which indicates that the simulated domain can be sufficiently modeled with the Unweighted Synergy Graph model. However, although the Unweighted Synergy Graph model has a similar performance, it contains 60 vertices compared to SGraCR's 12, showing that the Unweighted Synergy Graph model does not scale as well as the SGraCR model to more complex scenarios involving modular robots. Thus, the SGraCR model is well-suited for configurable robots in multi-robot teams.

7.5.4 Experiments with Real Robots

In our final set of experiments, we used Lego NXT robots in a pseudo-manufacturing setting. We chose the Lego platform as the hardware is modular and configurable to fit any task. We designed the robot task such that it involved manipulation and movement, which are essential components of many robot domains. Since we only used the time of task completion to train the SGraCR model, the approach in our experiments would be identical if any other robot platform or task was used. Figure 7.7a shows the layout of our robot experiments, and Figure 7.7b shows a NXT robot approaching L_1, with some of its components labeled. Each robot was programmed to follow a white line from station to station, and pass transparent cups to each other. The drilling/milling operations were not actually performed but assumed to take place either at the stations or by the robot transporting it.

Due to the limited carrying capacity of the NXT robots, we set the carrying capacity modules $M_2 = \{1\}$. Also, we had the physical limitation of 3 NXT robots, so teams had a maximum size



(a) (b)

Figure 7.7: a) The layout of the experiments involving NXT robots transporting items from L_0 to L_3. b) A NXT robot as it approaches L_1.

of 3. As such, $|\mathcal{T}| = 45$. In each trial, the robots moved 3 items from the start location L_0 to the end L_3, handing 1 item to each other at each station. The value of a team was the negative of the cost and the time taken, i.e., a team T with cost c and took t seconds to transport all 3 items had a score of $V(T) = -U_{\text{cost}}(c) - U_{\text{time}}(t)$. For these experiments, we set $U_{\text{cost}}(c) = c$ and $U_{\text{time}}(t) = t$, but the results should be representative of any utility function. We reduced the value of a team by its cost for two reasons: to show the efficacy of the SGraCR model over different value functions (compared to the previous subsections), and to better reflect that the cost of a team has an effect in a manufacturing scenario.

Similar to the experiment in simulation, we ran all 45 teams and computed their value, then scaled them to form each team's score. The 45 scores then formed then complete data set. We used a subset of the data set to learn a SGraCR. We then formed a multi-robot team using the learned SGraCR. In these experiments, $|O_{\text{train}}| = 20$, which is less than half of $|\mathcal{T}|$. The SGraCR and Synergy Graph models have 9 and 15 vertices respectively, and so we chose a training size of 20 to provide enough information to solve for the unknowns. We ran 100 trials, where each trial used a different subset of 20 training data. The *Real Robots* column of Table 7.4 shows the results of our robot experiments. SGraCR outperforms the Unweighted Synergy Graph and IQ-ASyMTRe approaches, demonstrating that SGraCR captures interactions that are unmodeled by the Unweighted Synergy Graph model and IQ-ASyMTRe. We performed a one-tailed paired T-test on the results, and found that SGraCR has a statistical significance of $p = 4 \times 10^{-7}$ against the Unweighted Synergy Graph model, and a statistical significance of $p = 8 \times 10^{-9}$ against IQ-ASyMTRe. Thus, the SGraCR model is robust and well-equipped to be applied to robot domains involving configurable multi-robot teams.

7.6 Robust Team Formation in a Foraging Task

We applied the Robust Synergy Graph for Configurable Robots (ρ-SGraCR) model to real robots in the foraging domain, to demonstrate its efficacy and relevance to real robot scenarios. We used three types of robot platforms: Lego NXTs, CreBots, and Aldebaran NAO humanoid robots. We chose these robots as they represent a spectrum from being easily reconfigurable (NXT) to being difficult to reconfigure (NAO). Figure 7.8 shows the three types of robots. The CreBots are iRobot Creates with TurtleBot hardware running our CoBot software. The colored squares on the NXT and CreBot are used for global localization. Within each platform, we defined configurable hardware modules for greater heterogeneity in the team (explained later).

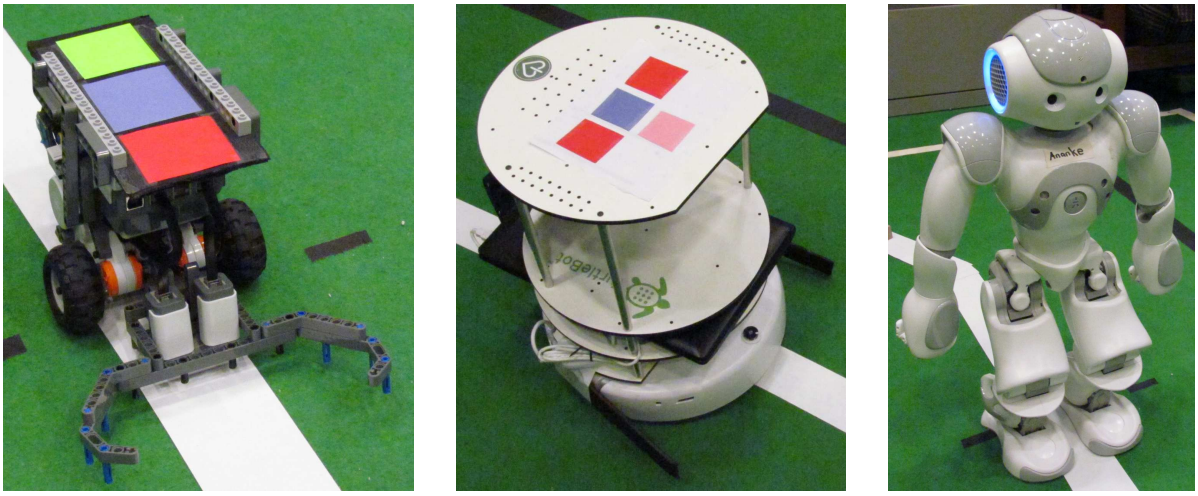
7.6.1 The Foraging Task

The task of the multi-robot team was to forage wooden blocks from a 4m $\times$ 3m area to two stockpiles located on each side. Figure 7.9 shows the setup of the experiment. There were 9 wooden blocks (resources to forage) in total, belonging to two types: colored (i.e., yellow, blue, and orange) and uncolored (i.e., regular brown). One of the blocks was located inside a small tunnel that was accessible only by NXTs (the other robots would not fit), and two of the blocks were placed on top of the tunnel and released only when a CreBot or NAO was nearby (the NXT was too short to activate the dropping mechanism). We set up these three blocks as "bonus" resources that can be foraged only when the right robot is included in the team.

The stockpiles on the left side of the field was for uncolored blocks, and the right for colored blocks. The robots had three minutes to complete the task, and their utility was:

$$\begin{aligned}\text{Utility} = U_{\text{g}} \cdot (|B_{\text{g,c}}| + |B_{\text{g,i}}|) + U_{\text{d}} \cdot (|B_{\text{d,c}}| + |B_{\text{d,i}}|) \\ + \sum_{b \in B_{\text{g,c}}} U_{\text{time}}(t_{\text{total}} - t_b) + \sum_{b \in B_{\text{d,c}}} U_{\text{time}}(t_{\text{total}} + t_{\text{drop}} - t_b)\end{aligned} \tag{7.8}$$

where U_g and U_d are the utilities for foraging blocks on the ground and blocks that were dropped respectively, and $U_{\text{time}}(t) = t$ is the utility function that converts time in seconds to utility values. The first subscript (g/d) of B indicates the initial position of the block (ground, dropped), and the second subscript (c/i) indicates if the block was foraged to the correct stockpile (correct, incorrect), e.g., a colored block at the colored stockpile is correct; a colored block at the uncolored stockpile is incorrect. Blocks foraged to the correct stockpile received a time bonus based on the time remaining (in seconds) when the block was foraged. Dropped blocks had an additional time bonus; coordinating to drop the blocks right before foraging them had higher utility.



(a) (b) (c)

Figure 7.8: The three robot platforms used in the foraging task: a) Lego NXT; b) CreBot; c) Aldebaran NAO.

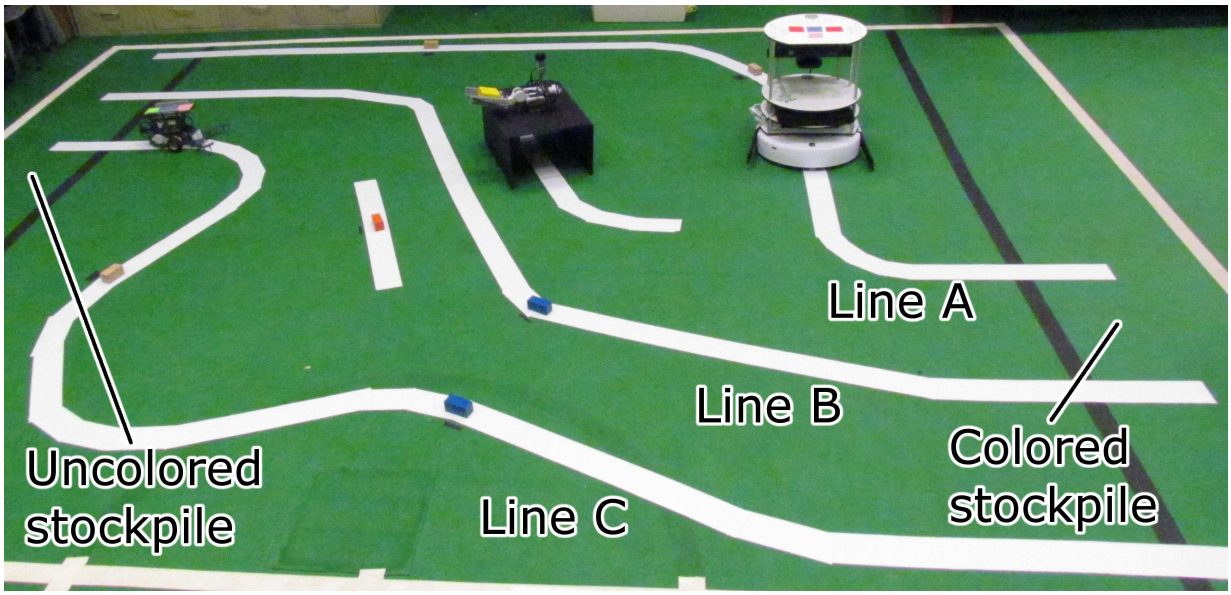


Figure 7.9: The setup of the foraging experiment showing the initial robot positions and wooden block positions. Uncolored and colored wooden blocks are to be foraged to their respective stockpiles on the left and right sides of the field.

The utility function was designed so that foraging any block would yield some utility, but foraging blocks to the correct stockpile had a large bonus (from the time bonus). Dropped blocks had extra utility to reflect the higher amount of coordination between the robots (one robot had to drop the blocks, and another to forage them).

7.6.2 Robot Types and Behaviors

We used three robot platforms (NXT, CreBot and NAO), and defined the modules as follows. $M_1 = \{\text{normal}^*, \text{fast}\}$ were the motors, where all platforms could use their normal motors, and the NXT had the option of faster motors. $M_2 = \{\text{mute}^*, \text{comm}\}$ was the communication modules, that allowed the robots to communicate and coordinate. $M_3 = \{\text{odometry-only}^*, \text{global-localization}\}$ was the localization module, that allowed the NXTs and CreBots to know their global (x, y) position (the NAOs did not have the global localization module). Lastly, to differentiate the robot platforms, $M_4 = \{\text{none}^*, \text{NXT}, \text{CreBot}, \text{NAO}\}$, where none indicated a failed/non-existent robot (explained later). The superscript * indicates that a module is the fall-back module of its type, i.e., it has a success probability of 1, and if another module fails, the robot uses the fall-back module.

Only two robots ($4 \times 2 = 8$ modules) performed the task at each trial. For example $T = \{$(fast, mute, global-localization, NXT), (normal, mute, odometry-only, NAO)$\}$ is a two-robot team with a NXT and NAO, where both robots could not communicate, and the NXT had fast motors and global localization while the NAO did not. Robot teams are able to communicate only if *both* of them have communication modules.

The behaviors of the robots depend greatly on the configuration of the team. Generally, the NXTs would perform line following to forage the blocks on the lines connecting the two stockpiles (A, B, and C in Figure 7.9), and would also forage the blocks at the disconnected lines if they had global localization. The CreBot and NAO robots would drop the blocks at the start of the trial; if they could communicate with the NXT, then they would coordinate with the NXT to maximize t_{drop} in Equation 7.8. The CreBot would forage the block closest to its initial location, and other blocks if it had global localization; the NAO does not forage any blocks. The robots did not know which blocks were colored or uncolored, unless a NAO was on the team and both robots had communication modules — the NAO robot provided a unique benefit, despite only being able to drop the blocks and not forage any blocks on its own.

We designed the foraging task, robot configurations and behaviors in such a way that we (as humans and the task designers) did not know upfront what the optimal team for the task would be. We wanted to test the efficacy of the ρ-SGraCR model in learning the robot team performance and forming a robust team in a complex task scenario.

7.6.3 Experimental Setup

We are interested in forming a robust multi-robot team by configuring the robot modules. We manually defined each module's failure rate for the experiment (the fall-back modules had 100% probability of success, and other modules had varying success rates from 30% to 80%). Since it is difficult to get robot modules to fail on demand at the desired failure rate, we instead ran the foraging trials assuming modules were always successful, and did the analysis of module failures separately. For example, suppose that a team T had a success probability of 0.7, and would become T' with probability 0.2 and T'' with probability 0.1. We used the utility attained by T, T' and T'' to define the ground-truth performance of T, i.e., with 0.7 probability of attaining Utility(T), 0.2 probability of Utility(T') and 0.1 probability of Utility(T'').

With the modules defined in the previous subsection, there were 14 unique robot configurations (8 NXTs, 4 CreBots, and 2 NAOs) with 84 feasible two-robot teams and 8 feasible one-robot teams. Teams were considered feasible if there was at least one NXT in the team. The goal was to form a robust two-robot team, but a two-robot team can become a one-robot team if the robot base module fails on one of them.

We performed 30 trials in the foraging experiments to enumerate all the feasible teams. Only 30 trials were necessary since the robot behaviors did not always change based on the module configuration, e.g., a team with both robots having no communication modules performs identically to a team where one robot has the communication module.

We then performed 10-fold cross validation, where 90% of the training data (utilities of teams) was used to learn a ρ-SGraCR model, and the learned model is used to form the risk-adverse team. We set the performance threshold to be 700 for the trials, which is slightly less than the mean utility attained by the teams. We used the *ApproxRobust* team formation algorithm, where $N_{\text{fail}} = 4$. We only used `ApproxOptimalRobustTeam` as the optimal algorithm `FormOptimalRobustTeam` is infeasible to be run on general problems due to its exponential runtime.

To compare the performance of our model and algorithm, we used two benchmarks. First, we used a *highest utility heuristic*, that computed the robustness score of the team that attained the highest utility (i.e., the team that had the best utility assuming no modules failed). Second, we used a *market-based technique* where each module bid using the utility attained from the training data and module failure probabilities:

$$\text{Bid}(m) = \frac{1}{\eta} \sum_{T \text{ s.t. } m \in T} \mathbb{P}(T) \cdot \text{Utility}(T)$$

where $\frac{1}{\eta}$ is a normalizing factor.

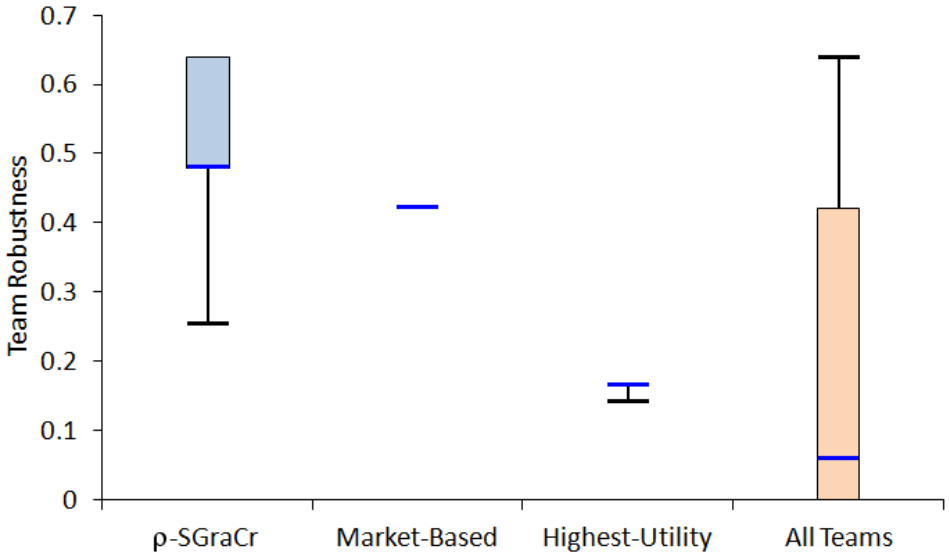


Figure 7.10: The robustness scores of teams formed by ρ-SGraCR and competing approaches. The dark blue line indicates the median, the top and bottom of the box represent the 75^{th} and 25^{th} percentiles, and the top and bottom whiskers represent the maximum and minimum values.

7.6.4 Experimental Results

Figure 7.10 shows the results of our robot experiments, where the dark blue line indicates the median, the top and bottom of the box represent the 75^{th} and 25^{th} percentiles respectively, and the top and bottom whiskers represent the maximum and minimum values. The robustness scores of all the teams were distributed between 0 (worst team) and 0.64 (optimal team), with a median of 0.06. ρ-SGraCR formed a team with a median robustness of 0.48, with the 25^{th} percentile also at 0.48. The 75^{th} percentile and maximum value are 0.64, which is the robustness of the optimal team of this experiment; ρ-SGraCR found the optimal team in 4 of the 10 trials. The market-based algorithm always found the same team that had a robustness score of 0.43. ρ-SGraCR outperformed the market-based algorithm in 8 of the 10 trials (the other 2 trials had robustness scores of 0.42 and 0.26). The highest-utility heuristic, i.e., picking the team that attained the highest utility, had a score of 0.17 in 9 of the 10 trials and 0.14 in 1 trial.

Thus, the ρ-SGraCR model outperforms the competing market-based algorithm and highest-utility heuristic (with p-values of 0.025 and 5×10^{-6} respectively using a one-tailed paired Student's t-test), and found the optimal robust team 40% of the time. In 9 of the 10 trials, the team formed is above the $75th$ percentile, which reflects that the ρ-SGraCR model effectively modeled the team performance and formed good robust teams. As such, our results demonstrate that ρ-SGraCR is well-suited to model the performance of robust multi-robot teams in challenging scenarios where the optimal team is difficult to compute *a priori*.

7.7 Chapter Summary

This chapter presented extensive experiments in simulation and on real robots, where the Synergy Graph model was learned from data, and applied to actual problems:

- We compared the Unweighted Synergy Graph with ASyMTRe when the robot capabilities follow a probabilistic model [Parker and Tang, 2006] and showed that the learned Unweighted Synergy Graph formed teams with effectiveness of at least 0.93 compared to at most 0.64 with ASyMTRe.
- We learned Unweighted and Weighted Synergy Graphs using the RoboCup Rescue simulator [RoboCupRescue, 2011], and outperformed IQ-ASyMTRe with a statistical significance of $p = 5 \times 10^{-137}$ with 4 agents, and $p = 4 \times 10^{-9}$ with 5 agents.
- We learned WeSGRAs using the RoboCup Rescue simulator to perform role assignment, and showed through cross-validation that the learned WeSGRA has high log-likelihood. We demonstrated that the role assignments generated using the WeSGRA has high value, compared to a market-based approach.
- We used NAOs and NXTs in a foraging task, and showed that the role assignments formed using the learned WeSGRA outperforms a market-based approach, and is statistically significant to $p = 3 \times 10^{-5}$.
- In a manufacturing task, with simulated robots and real robots (NAOs and NXTs), we demonstrated that the Synergy Graph for Configurable Robots (SGraCR) model forms effective configurable robot teams, and outperforms IQ-ASyMTRe. Our results for the real robots are statistically significant to $p = 8 \times 10^{-9}$.
- The NAO, CreBot, and NXT robots performed a foraging task, and the goal was to form a robust multi-robot team. We showed that the team formed from using the learned Robust Synergy Graph for Configurable Robots (ρ-SGraCR) model is effective and forms the optimal team 40% of the time, outperforming a market-based approach and a highest-utility heuristic with $p = 0.025$ and $p = 5 \times 10^{-6}$ respectively.

The results in this chapter empirically show that the Synergy Graph is effective at modeling team performance in a variety of real robot problems, and outperforms competing approaches. Since the Synergy Graph algorithms only require observations of team performance as input, they are applicable to a wide range of multi-robot problems.

Chapter 8

Related Work

This chapter presents a review of related work, discussing the relevant domains of task allocation, role assignment and coalition formation. The discussion also covers the fields of ad hoc robot coordination, team formation, and operations research.

8.1 Multi-Robot Task Allocation

Multi-robot task allocation (MRTA) is focused on the problem of allocating a set of tasks T to a group of robots R so as to maximize a utility function. A thorough overview of this domain is given in [Gerkey and Mataric, 2004], and the authors categorize the problem along three axes: single-task robots (ST) versus multi-task robots (MT), single-robot tasks (SR) versus multi-robot tasks (MR), and instantaneous assignment (IA) versus time-extended assignment (TA). ST and MT define whether robots are capable of executing a single task (ST) or multiple tasks (MT) simultaneously. SR and MR define whether a task $T_j \in T$ can be completed with single robots (SR) or require multiple robots (MR). IA means the tasks are known and assigned once to the robots without future allocations, while TA means that tasks may arrive over time and assignments of tasks to robots have to be done as the tasks arrive.

This thesis considers ST-MR-IA, with the key difference that we consider forming a team for a single task, while MRTA considers the allocation of multiple teams to multiple tasks.

Representing Heterogeneous Capabilities

The capabilities of heterogeneous robots have been represented in different ways. Capabilities have been modeled as values, where higher values indicate better task performance [He and Ioerger, 2003], and with a Normal distribution to represent the uncertainty in the agents' perfor-

mance [Guttmann, 2008]. A general utility function has also been proposed [Gerkey and Mataric, 2004], where $U_{i,j} = Q_{i,j} - C_{i,j}$ is the utility attained if R_i is capable of performing T_j, $Q_{i,j}$ is the quality of the task execution, and $C_{i,j}$ is the expected resource cost. If R_i is not capable of performing T_j, then $U_{i,j} = 0$. The utility of a robot performing a task is quantified, and it is general and does not impose any restrictions on the utility function. However, this generality is a double-edged sword and does not impose any structure on the tasks that would allow them to be grouped.

Another technique models the number and types of resources that the robots possess. Let S be a set of resources, e.g., robot arm, camera, sonar [Shehory and Kraus, 1998, Chen and Sun, 2011]. Then, each robot R_i is defined as a vector $(s_{i,1}, \ldots, s_{i,|S|})$, where $s_{i,k}$ is the number of the resource S_k that R_i possesses. In this model, resources are either completely interchangeable (i.e., a robot arm on R_1 is equivalent to a robot arm on R_2) [Service and Adams, 2011a, Tosic and Agha, 2004, Shehory and Kraus, 1998], or with a level of quality, i.e., $q_{i,k} \in [0, 1]$ is the quality of each of the $s_{i,k}$ resources of type S_k in R_i [Chen and Sun, 2011]. In these resource models, tasks are defined as a list of required resources, e.g., a task requires 2 robot arms, 3 cameras, and 1 sonar, and in the latter model, a minimum quality $q_{min,i,k} \in [0, 1]$ of each resource is also defined.

The service model of capabilities [Service and Adams, 2011a, Vig and Adams, 2007] is similar to the resource model, with the following distinctions: a robot R_i is either able or unable to performance a service S'_k, i.e., there is no quantity or quality of a single service a robot possesses ($s'_{i,k} \in \{0, 1\}$). Tasks are defined as a list of the number of each type of service required (similar to the resource model), but the in task allocation process, each robot can only perform a single service, instead of providing all the services it is capable of. The authors state that while resources can be exchanged in multi-agent systems, resources on multi-robot systems are usually physical and cannot be exchanged. Having a minimum number of resources does not necessarily imply task completion because other constraints such as proximity have to be represented and met. The resource model can still be applied with the addition of constraints, but the service model abstracts and represents these constraints succinctly.

Robot capabilities have also been defined as a set of schemas [Parker and Tang, 2006, Tang and Parker, 2007, Zhang and Parker, 2010, Zhang and Parker, 2012], where schemas have defined inputs and outputs. Robots are defined as sets of schemas: the sensors of the robots provide outputs without any inputs, and other schemas are perceptual (e.g., vision, localization), motor (e.g., controlling actuators), and communication. The task is defined as a set of desired outputs, and a team of robot $R' \subseteq R$ is capable of completing the task if a joint plan exists for R' that produces the outputs by chaining the schemas in R'.

This thesis considers robot capabilities as Normal variables, where the mean corresponds to the peak performance and variance represents the dynamics of the world. In addition, the performance of a team is not just the sum of single-robot capabilities. Prior research has assumed that the capabilities of the robots are known a priori. This thesis learns the robot capabilities and team synergy from observations.

Evaluating Task Allocations

Let $T' \subseteq T$ be the subset of tasks that are allocated, $R_{T_j} \subseteq R$ be the robots allocated to perform task $T_j \in T'$, and $C_{i,j}$ be the cost of R_i executing T_j. There are two main methods to quantify the performance of an allocation of tasks to robots. The first method, task-based performance, defines a utility V_j gained by completing each task T_j. The second method, team-based performance, defines the quality of performance of each robot R_i performing the task T_j, i.e., $Q_{i,j}$.

Task-based performance is commonly seen in market-based techniques [Dias, 2004, Dias and Stentz, 2002] and other approaches [de Weerdt et al., 2007]. Each task is associated with a reward, that is computed based on domain-specific factors, such as the amount of unexplored space [Zlot et al., 2002] or number of items picked up [Dias and Stentz, 2000]. In market-based techniques, the cost $C_{i,j}$ of R_i performing T_j is used to form the bid, and an auctioneer assigns tasks based on robots' bids [Tang and Parker, 2007]. The final performance of the task allocation is then computed based on the profit, i.e., the difference between the utility gained from the tasks and the costs incurred: $\sum_{T_j \in T'}(V_j - \sum_{R_i \in R_{T_j}} C_{i,j})$ [Dias and Stentz, 2000]. In some approaches, only the sum of utilities gained is used to calculate performance, and the costs are only used for the allocation process, i.e., $\sum_{T_j \in T'} V_j$ [Vig and Adams, 2006a, Tang and Parker, 2007], and in others the goal is to complete all the subtasks while minimizing the total cost [Lim et al., 2009]. The benefit of task-based performance is that the utility gained from completing a task is independent of how the task is completed — after the allocation, each task is either completed or not completed, and there is no measure of how well a particular task was done, other than the costs incurred by the robots assigned to it.

In team-based performance, the quality of a completed task varies depending on the robots allocated to it, e.g., R_1 may complete task T_1 with a lower quality than if R_2 completed T_1. The performance of the task allocation is then the difference between the quality of completed tasks and the costs incurred by the robots: $\sum_{T_j \in T'} \sum_{R_i \in R_{T_j}} (Q_{i,j} - C_{i,j})$. With this formulation, the ST-SR-IA problem can be posed as an optimal assignment problem and solved in $O(mn^2)$ time, or with a market-based approach where the bids are $Q_{i,j} - C_{i,j}$ and the highest bid wins the auction [Gerkey and Mataric, 2004]. When each task requires more than one robot to complete, it becomes a ST-MR-IA problem, and has many similarities with the coalition formation problem,

which we describe later. Team-based performance measures how well a task was completed, so the composition of a team has a larger impact beyond the costs. The ST-MR-IA problem is strongly NP-hard, and heuristics have been considered to solve the problem, that also account for inter-task constraints [Zhang and Parker, 2013].

This thesis uses team-based performance, where the composition of the team has a large effect on the team performance. However, the team performance is beyond the sum of individual capabilities, and is dependent on the Synergy Graph structure that describes the task-based relationships among the robots.

8.2 Role Assignment

In role assignment, there is a single task that can be divided into a set of roles $\mathcal{R}$. Each role $\mathcal{R}_j$ specifies the behavior of the robot performing it [Stone and Veloso, 1999], and the role assignment problem is to find the optimal assignment of roles to robots. Role assignment has many similarities with ST-SR-IA and ST-SR-TA of MRTA. One key difference is that while tasks in MRTA are typically independent, roles can have interdependencies that affect the overall performance of a team of robots. In addition, while the set of roles $\mathcal{R}$ is fixed, formations can be used to define the active roles at a point in time, with triggers to switch between formations as needed [Stone and Veloso, 1999]. Similarly, in the Skills, Tactics and Plays (STP) paradigm, plays determine which roles are currently active and available to be assigned to robots, and applicability conditions are used to switch between plays over time [Bowling et al., 2003, Browning et al., 2005].

Role assignment has been used in many domains, such as robot soccer [Roth et al., 2003, McMillen and Veloso, 2006], agent organizations [Le et al., 2011, Dastani et al., 2003], treasure hunt [Jones et al., 2006], assembly [Simmons et al., 2000], and formation control [Chen and Wang, 2007, Ji et al., 2006].

Market-based techniques are also frequently used for role assignment, where robots submit bids on roles based on their state, and roles are assigned to the highest bidder [Frias-Martinez et al., 2004, Jones et al., 2006]. Regions are also used in role assignment, where a robot that is already in a role's region continues to take that role, in order to minimize role switching among the team [McMillen and Veloso, 2006]. In order to synchronize the available roles in the team, a shared world model is used, where robots periodically update their teammates with their observations and state, so that every robot has a similar world model [Vail and Veloso, 2003, Coltin et al., 2010, Liemhetcharat et al., 2010].

In many role assignment domains, robots are assumed to be homogeneous or capable of being assigned any role [Browning et al., 2005]. The state of the team is then the key factor used to choose between role assignments [Chen and Wang, 2007]. In heterogeneous teams, the capabilities of the robots have to be considered for effective role assignment, and the capabilities can be represented as a binary for each possible role, i.e., $Cap_{i,j} = 1$ iff R_i can perform role $\mathcal{R}_j$ [Zhu and Alkins, 2009].

However, while different role assignment algorithms have been proposed and analyzed, there has been relatively little research done in the formal definition of roles, other than a description of its behavior [Stone and Veloso, 1999] or a mapping of roles to robots that are capable of fulfilling it [Zhu and Alkins, 2009]. In particular, roles have been treated as atomic and not split into relevant components to better describe its purpose in a heterogeneous team.

This thesis considers role assignment with the Weighted Synergy Graph for Role Assignment (WeSGRA) model, that is an extension of the Synergy Graph model where the agents have multiple capabilities, one for each role. The robots are heterogeneous, and we model the synergistic effects of the robots in the role assignments.

8.3 Coalition Formation

Coalition formation involves the partitioning of a set of agents A into disjoint subsets so as to maximize the overall utility. In characteristic function games (CFGs), the value of each possible subset $S \subseteq A$ is given by a characteristic function V, and the goal is to find the optimal coalition structure CS to as to maximize $\sum_{S \in CS} V(S)$, where $\forall S_i, S_j \in CS, i \neq j \Rightarrow S_i \cap S_j = \emptyset$, and $\bigcup_{S \in CS} S = A$ [Sandholm et al., 1999]. The number of coalition structures is $O(|A|^{|A|})$ and $\omega(|A|^{|A|/2})$ [Sandholm et al., 1999], so enumerating possible coalition structures to find the optimal is intractable. Coalition formation is applicable to MRTA [Service and Adams, 2011a] in domains such as deciding flight targets for UAVs [George et al., 2010].

Coalition formation focuses on how to partition a set of agents to maximize a value function, while this thesis is interested in modeling the value function, based on noisy observations of the robots at the task.

Establishing Bounds on Coalition Structures

To find a coalition structure CS within a bound from the optimal CS^*, i.e., $k = \min\{\kappa\}$ where $\kappa \geq \frac{V(CS^*)}{V(VS)}$, the minimum number of coalition structures visited is $2^{|A|-1}$ and the bound is $k = |A|$ [Sandholm et al., 1999]. In addition, the authors provided an anytime algorithm that

lowers the bound, by representing all coalition structures in a tree, searching the bottom two levels of the tree (with $2^{|A|-1}$ nodes), and then searching from the top of the tree downwards through the layers, with the ability to stop at any time once the bottom two layers have been searched.

An anytime algorithm has been designed that finds a coalition structure within a factor k of the optimal [Service and Adams, 2011b]. Coalition formation has been applied to task allocation [Service and Adams, 2011a], where the CFG is defined to be based on the task a coalition is assigned to complete, i.e., the task-based performance model of MRTA. They provided two algorithms that are capable of finding a coalition structure within a factor of $k + 1$ of the optimal, assuming that coalitions are restricted to k or fewer agents. The first algorithm uses a resource-based model for task allocation and is able to complete within $O(|A|^k m)$ time, where m is the number of tasks. The second uses the service-oriented task allocation model and runs in $O(|A|^{3/2} m)$ time. Further, they showed that if there are p types of agents, then a dynamic programming-based algorithm can find the optimal coalition structure in $O(|A|^{2p} m)$ time. If individual coalition values can be observed, instead of the value of a coalition structure, then dynamic programming can solve the coalition formation problem in $O(3^{|A|})$ time [Vig and Adams, 2007]; however, this is infeasible when a coalition's value can only be evaluated when it is paired with a task.

Forming the optimal team is NP-hard, and this thesis contributes two team formation algorithms: one that forms the δ-optimal team in exponential time, and one that approximates it in polynomial time. This thesis showed that the approximation algorithm forms near-optimal teams with high performance in the extensive experiments on real robots.

Techniques to Form Coalitions

Besides the algorithms that have bounded guarantees listed above, other techniques have also been used to form coalition structures. Coalition formation has also been applied to task allocation using the task-based performance model [Vig and Adams, 2006b]. However, they use a heuristic function to form the coalitions, in order to balance between computation and communication, as well as coalition imbalance, which is how dependent a coalition is on a single member providing a large amount of the required resources of the task.

Constraints have been used to verify the feasibility of robots performing tasks even if the resource requirements are met [Vig and Adams, 2007]. They defined a metric called FTC that trades-off between the imbalance in a coalition and its size. In addition, the authors cast the multi-robot coalition formation problem as winner determination in combinatorial auctions, where the bidders are tasks, items are robots, and bids are the utility that each task has to offer for a par-

ticular subset of robots. Winner approximation in combinatorial auctions is inapproximable, and the authors provide a market-based bidding process in order to solve this problem in a distributed manner.

Since coalition formation is NP-hard, many heuristic solutions have been proposed, but it is often difficult to pick the right heuristic for a problem. The intelligent Coalition Formation for Humans and Robots (i-CiFHaR) framework was recently introduced, that models the features of coalition formation problems and utilities of coalition formation algorithms in a decision network [Sen and Adams, 2013].

This thesis compares the Synergy Graph algorithms against competing approaches, including market-based techniques. One difference between the Synergy Graph model and i-CiFHaR is that the latter learns a utility value for every combination of coalition formation problem features, causing the utility table to be exponential in size; Synergy Graphs model the synergistic effects of agents, and would not have an exponential size.

Externalities in Coalitions

In characteristic function games, the value of a coalition depends only on the agents within it. There has been recent work in coalition formation with externalities, where a coalition's value depends on the structure of other coalitions. A logic-based representation is used for coalition formation with externalities, that is fully expressive and at least as concise as regular coalition formation representations [Michalak et al., 2010]. Positive and negative externalities can be considered separately, where PF^{+}_{sub} means weakly sub-additive, so merging two coalitions decreases their joint value (or keeps it constant) while increasing the values of other coalitions in the structure (or keeps them constant), and PF^{-}_{sup} means weakly super-additive, where merging a coalition increases its value and decreases the values of other coalitions (or keeps values constant) [Rahwan et al., 2009]. The authors compute the upper and lower bounds for coalition values in these settings, and identify the minimum search to establish a bound k from the optimal coalition structure.

Mixed externalities have also been considered, where both positive and negative effects can occur [Banerjee and Kraemer, 2010]. They define agents with a set of type $\mathcal{T}$, and that agents of some types as competitors, and the value of a coalition structure CS improves if they are in singleton coalitions. Conversely, agents of the remaining types are collaborators, and the value of CS improves if all of them are in a single large coalition. Using this formulation, the authors use a branch and bound algorithm to find the best coalition structure with guaranteed worst-case bounds.

Externalities in coalitions is an interesting area because it considers how the values of coalitions are computed. In regular coalition formation, the characteristic function defines the values of coalitions, and the function is assumed to be general and unknown, so little work has been done to analyze possible structures in the characteristic function.

Externalities model how coalition values change based on the composition of other coalitions, this thesis is interested in how the value of a team (or coalition) is formed based on the members within it, and not those outside it.

Other Formulations

In addition to externalities, coalition formation under uncertainty has been considered, where agents have different types and skill levels (good, average, bad) [Chalkiadakis and Routilier, 2010]. The skills levels affect the probability of success of actions, e.g., building an outstanding house with a good painter, good carpenter and good builder has a high probability of success, while doing so with a bad painter/carpenter/builder has a low probability of success. The agents know their types, but not the types of other agents, and they have to update their beliefs of agents they have interacted with through observations of interactions. Thus, agents have to trade off between exploring new teammates versus exploiting current knowledge about previous teammates.

Coalition Skill Games (CSGs) are also considered, where there is a set of skills, and each agent has a subset of skills [Bachrach et al., 2010]. Then, a coalition of agents can perform a task only if the union of skills is a superset of the skill requirement of the task. The authors showed that any coalition formation problem can be cast as a Weighted Task Skill Game, and considered the case where all weights are positive, in order to find the optimal coalition structure in polynomial time.

Coalition formation with spatial and temporal constraints were explored [Ramchurn et al., 2010], and the main domain was in urban search-and-rescue. The authors proved that the problem is NP-hard, and provided a heuristic algorithm to solve it. Each task in the domain required a number of units of work, and increasing the number of agents on a task decreased the amount of time required to complete the task.

This thesis models capabilities of robots with Normal variables, and computes the team performance of a multi-robot team using the capabilities and the task-based relationships among the robots.

8.4 Ad Hoc Teams

The ad hoc domain was recently introduced, and the goal is "to create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of contributing as team members" [Stone et al., 2010]. An example of an ad hoc problem is with two robots — one that is pre-programmed to follow a certain policy, and another that has to adapt its behavior so that the pair will be jointly optimal without explicit communication. The pre-programmed robot and the ad hoc robot can be viewed as the teacher and learner in a multi-armed bandit problem [Stone and Kraus, 2010, Barrett and Stone, 2011]. Similarly, a two-player game-theoretic setting is used to study how an ad hoc agent can vary its actions so as to maximize the payoff with a best-response teammate that has varying amounts of memory of previous interactions [Stone et al., 2009], with extensions to situations where a single ad hoc agent leads multiple teammates in selecting the optimal joint action [Agmon and Stone, 2012].

Role assignment in an ad hoc team is considered [Genter et al., 2011], where an ad hoc agent has to select a role, such that it maximizes the team's overall utility based on its observations of its teammates. An ad hoc agent in the pursuit domain has to vary its behavior to better suit the team's objective of capturing the target, by modeling its teammates and choosing a best response [Barrett et al., 2011]. In the case where the system state and joint action is fully observable, but the model of teammates is unknown, biased adaptive play can be used by an ad hoc agent to optimize the joint action of the team [Wu et al., 2011].

Ad hoc agents have been applied to flocking [Genter et al., 2013], where the ad hoc agent adjusts its heading to direct the flocking agents. Different strategies for ad hoc agents have been proposed, to improve performance when the teammates are Markovian [Chakraborty and Stone, 2013], and when their behavior models can be learned [Barrett et al., 2012].

Locker-room agreements, i.e., policies agreed upon by the team prior to evaluation, can be used to coordinate robots without additional communication. Robots can estimate the state of a teammate in order to decide which robot should approach the ball in the robot soccer domain [Isik et al., 2006], and the authors showed that while the robots agree on the policy $> 90\%$ of the time, communication is necessary for situations where errors in state estimation may cause policy fluctuations in the team.

Research in the ad hoc domain has so far focused on how an ad hoc agent can adapt to its teammates in order to improve task performance. This thesis considers how to form an effective team comprised of learning robots, that progressively learn to coordinate better together with experience.

8.5 Team Formation

Team formation is focused on the problem of selecting the best subset $R^* \subseteq R$ that can complete a task T. Robots can be defined with schemas [Parker and Tang, 2006] (described above) and the team R^* is selected by composing feasible teams through planning, and then selecting the optimal one using a heuristic in the ASyMTRe algorithm. ASyMTRe has been extended to form teams that complete multiple tasks, using both team formation and a market-based task allocation algorithm [Tang and Parker, 2007]. The skills and scores of agents in a team can be tabulated, and solved with a linear program to form the best team [Boon and Sierksma, 2003].

Graphs can be used to represent relationships among agents, where a subgraph of connected agents are selected to complete a task [Gaston et al., 2004, Gaston and desJardins, 2005]. Similarly, by using social networks of agents, where agents have different skills, and edge weights represent communication costs, the optimal team to complete the task has to cover all the required skills [Lappas et al., 2009, Li and Shan, 2010], or trade off between skills and connectivity [Dorn and Dustdar, 2010]. The edges in a social network graph can also be used as constraints, where an agent is assigned a task, and must find teammates that are directly connected to it [de Weerdt et al., 2007], or form a connected sub-network [Bulka et al., 2007].

This thesis introduces the Synergy Graph model, that is motivated by social graphs, but adapted to robots executing a task. In particular, this thesis models the capabilities of the robots and how team performance varies as a function of the team composition. Also, this thesis compares the performance of the Synergy Graph against ASyMTRe.

8.6 Operations Research

In operations research, agents and resources are represented in a matrix K, where $K_{i,j}$ represents the knowledge that the i^{th} agent knows about the j^{th} resource. The goal is then to obtain a diagonalization of K in order to form team of agents and resources, subject to some measure of performance [Agustin-Blas et al., 2011, Dereli et al., 2007, Zakarian and Kusiak, 1999]. A similar approach is used to form cross-functional teams by selecting members from different departments of a company, where individual and pairwise performance are considered for various criteria, and each member is given a score for every criteria. In such a formulation, the goal is to jointly optimize individual performance, pairwise performance and exterior organizational collaborative performance, and a genetic algorithm is used to solve the team formation problem [Feng et al., 2010]. While pairwise interactions are considered, it is assumed to be given by experts and not learned.

Another aspect of team formation in operations research considers the composition of a team during the execution of a task, such as soccer. One method is to find the optimal team formation (e.g., the number of strikers and mid-fielders) throughout the game, considering the current score and time remaining [Hirotsu and Wright, 2003]. Another approach is to consider the overall team strategy, such as whether to play defensively or aggressively, and to choose between non-violent and violent styles of play (a violent style of play is more risky and commits more fouls, but has a higher likelihood of scoring goals) [Dobson and Goddard, 2010]. However, these approaches are focused on adjusting the team as a result of the state of the game, and not the optimal selection of a line-up of players on the team. Forming the optimal line-up involves assigning scores to players and positions based on skills such as passing and keeping a ball. These scores are computed with a linear program and the score of a team is the sum of scores of players in positions [Boon and Sierksma, 2003].

Most team formation algorithms in operations research assume that the capabilities and knowledge of agents are given by an expert, and focus on how to best allocate personnel to tasks. This thesis is interested in representing the capabilities and performance of the team, learning the capabilities from observations, and then forming an effective team. The learning approaches in operations research consider team performance as a linear function of individual capabilities, while this thesis considers the synergistic effects of members in a team.

8.7 Robustness and Redundancy

Robustness in multi-robot systems can be considered with three areas: detecting when robots have failed, diagnosing and identifying robot failures, and responding to robot failures [Parker, 2011]. Research in multi-robot robustness has mostly focused on coordination algorithms that are robust to failures. For example, coordination strategies using self-organization and self-healing are applied to resource-flow systems in order to complete a task even with robot failures [Parker, 1998, Preisler and Renz, 2012]. A distributed algorithm monitors key agents in a team to increase its failure-detection and robustness [Kaminka and Tambe, 2000], and a robust multi-robot data association technique improves performance in SLAM [Cunningham et al., 2012]. We are interested in forming a robust multi-robot team where the behaviors and coordination strategies of the robots are pre-defined — our approach forms the team that maximizes robustness given such algorithms.

Redundancy has been used to form robust multi-robot and multi-agent teams. Robots are modeled as Markov processes and redundancy allows the team to stay in desired states when some robots fail [Napp and Klavins, 2010]. Similarly, multiple software agents are run con-

currently to procure a service, so that at least one of the agents completes the task before the deadline [Stein et al., 2011]. We model and consider how the performance of the team varies with possible failures, in order to find a robust team. We do not assume that redundancy increases robustness — we show how our model also handles scenarios where redundant robots decrease overall team performance.

Research in robust robot teams has mostly focused on coordination algorithms that allow robot teams to adapt to failures among the teammates. This thesis considers forming a robust team that maximizes the probability of attaining a threshold performance in the presence of failures.

Chapter 9

Conclusion and Future Work

This chapter presents the scientific contributions of this thesis, and presents several interesting directions for future work that build on this thesis.

9.1 Contributions

This thesis makes the following scientific contributions:

- **Synergy Graph Model**
  We formally defined the Synergy Graph model, where agents are vertices in a connected graph, and their *task-based relationships* are modeled with the edges of the graph. The agents' capabilities are modeled with Normal variables that capture the inherent variability in performance in a dynamic world. We further defined *pairwise synergy* and *synergy* that uses a Synergy Graph to compute the performance of a multi-agent team. The Synergy Graph model represents team performance beyond the sum of the capabilities of the members of the team; the *synergy* among the team members play a large role in the team performance.

- **Two Team Formation Algorithms**
  We presented two team formation algorithms, `Form`δ`OptimalTeam` and `Approx`δ`OptimalTeam`, that use a Synergy Graph to form a multi-agent team. `Form`δ`OptimalTeam` uses branch-and-bound to form the δ-optimal team in exponential time, and the second algorithm, `Approx`δ`OptimalTeam`, uses simulated annealing to approximate the δ-optimal team in polynomial time. We showed that `Approx`δ`OptimalTeam` forms near-optimal teams without exploring a large amount of the search space.

- **Synergy Graph Learning Algorithm**
  We contributed `LearnSynergyGraph`, the algorithm that learns a Synergy Graph from observations of team performance. The algorithm iteratively improves the Synergy Graph structure and learns the agents' capabilities using either a least-squares solver or a non-linear solver. We showed that `LearnSynergyGraph` learns Synergy Graphs with high log-likelihood, and the learned Synergy Graphs can be used to form effective teams.

- **Synergy Graph Iterative Learning Algorithm**
  We presented `AddTeammateToSynergyGraph`, a learning algorithm that adds a new teammate into an existing Synergy Graph. The algorithm uses heuristics to generate an initial guess of the new teammate's edges in the Synergy Graph structure, and iteratively improves the edges and learns the new teammate's capabilities. We showed that the `AddTeammateToSynergyGraph` algorithm can be run iteratively to add new teammates.

- **Extensions to the Synergy Graph Model**
  We showed that the Synergy Graph model is extensible to capture other characteristics, by considering modifications such as agents with multiple capabilities (to apply Synergy Graphs to role assignment problems), graphs with multi-edges (to model configurable robots), and non-transitive task-based relationships.

- **Two Robust Team Formation Algorithms**
  We presented how the Synergy Graph model is augmented to model agents that probabilistically fail, and showed that the augmented Synergy Graph models situations where redundancy improves robustness, and where redundancy reduces robustness. We contributed two robust team formation algorithms, `FormOptimalRobustTeam`, that forms the optimal robust team in exponential time, and `ApproxOptimalRobustTeam`, that approximates the optimal robust team in polynomial time.

- **Team Formation with Learning Agents**
  We detailed how Synergy Graphs represent agents that learn to collaborate better over time, using edges that decrease their weight, and how such a representation models ad hoc agents and other learning agents. We contributed an algorithm that uses heuristics from the multi-armed bandit problem to allocate training instances in order to maximize the performance of the multi-agent team formed after training, and showed that the upper confidence bound heuristic performed the best with the lowest regret compared to the optimal.

- **Empirical Evaluation of the Synergy Graph Model** We applied the Synergy Graph model to various simulated and real robot problems, using simulated rescue robots, and real robots such as the Aldebaran NAO humanoids, Lego Mindstorms NXTs, and CreBots. We demonstrated the the Synergy Graph model effectively captures the team performance of heterogenous robots in dynamic problems, and outperforms competing approaches.

9.2 Bridging Previous Work and Future Work

During the course of my undergraduate and Ph.D. studies, I worked on several robot projects that helped to shape my research direction and thesis topic, most notably working on RoboCup from 2006. In this section, I describe some of the lessons learned from my RoboCup work, and how my thesis forms a bridge between my previous work and my future research directions.

- **A capable teammate improves team performance**
  In 2006 and 2007, I participated in the RoboCup 4-Legged League using the Sony AIBOs. The AIBOs played four-on-four games of robot soccer, where each team had one goalkeeper and three field players. I designed the behavior of the goalkeeper, whose main role was to defend against opposing robots who were trying to score goals.

  The outcome of a game of robot soccer depends on the number of goals scored for and against the team. By having a competent goalkeeper that effectively defended the goal, we drastically reduced the number of goals scored against us. Further, by clearing the ball out of the penalty box quickly, the field players were able to launch a counter-attack and score goals.

  Thus, improving the capability of a single member of the team increases team performance, and this aspect is captured in the agent capabilities of the Synergy Graph model. A future direction would be to analyze and model *how* a teammate's capability can be modeled beyond a Normal distribution, e.g., being state-dependent.

- **A capable teammate compensates for weaknesses in the team**
  We have observed over the years at RoboCup that with a capable attacker, i.e., a robot that approaches the ball and kicks it towards the opposing goal quickly and accurately, the other robots of the team contribute very little to the team performance. In fact, in the early games of the RoboCup competition, the goalkeeper does not have much to do since the ball is almost always on the opposing half of the field. The goalkeeper and other members of the team contribute more in the later games where opponents are increasingly challenging.

Similarly, a capable defender robot alleviates the need for a capable goalkeeper. The defender robot is designed to prevent the ball from going past a certain point in the field, that we refer to as the defending line. When the defender robot performs well at its role, the ball never gets past the defending line and the goalkeeper again contributes little to the team performance, whether or not the goalkeeper is highly capable.

These examples provided motivation that team performance is not just the sum of single-robot capabilities — the team performance in the examples above would be similar irregardless of the capabilities of the goalkeeper. The Synergy Graph model in this thesis considers task-based relationships in addition to robot capabilities, and a future direction would be to consider how certain robots can "mask" the effects of others on the team.

- **Teamwork contributes greatly to team performance**

  Our RoboCup team is divided into a number of roles: a goalkeeper, an attacker, an offensive supporter and a defensive supporter (or defender). The latter three roles are assigned dynamically to the three field robots, i.e., robots on the field that cannot enter the defensive penalty area. The role assignment of the field players is performed by using a shared world model. The robots communicate constantly to share their global position and the ball's global position. A distributed role assignment algorithm then runs in each robot, that uses the world model to compute its role. If communication is error-free and instantaneous, all robots on the team have the same world model and there is no conflict in the role assignment.

  The attacker role is the most important, as it is the only robot that approaches the ball to kick it into the opposing goal. In certain situations where there is latency or errors in communication, multiple robots may take on the attacker role (they cannot distinguish between failed communication and having all teammates switched off), and so multiple robots on the same team approach the ball. Such a phenomenon is similar to children playing soccer where everyone heads to the ball, and as such, no one is able to properly kick and score goals. Thus, having teamwork trumps having many individually capable robots.

  Another example of the contribution of teamwork comes from games against faster opponents. Certain teams in RoboCup focus their research on locomotion and faster walking gaits. Their individual robots can walk much faster than ours and typically reach the ball before our robots do. However, we developed effective positioning of the supporter robots (i.e., the non-attacker field robots) to overcome the speed difference. By placing a defensive supporter behind the attacker, if an opponent kicked the ball past our attacker, our

defensive supporter was already in position to receive the ball and kick it further up-field or towards the opposing goal. As a result, our team with better teamwork performed better than some teams that had faster walks but poorer teamwork.

Thus, I was motivated by the fact that a team of highly-capable individual robots may perform poorly compared to a team with individually less capable robots but with better teamwork, and this led me to develop the Synergy Graph model. The Synergy Graph model in this thesis considers pairwise interactions, and a future direction would be to consider state-dependency and interactions beyond pairwise relationships.

- **Slight differences in robot hardware have large effects**

  In the RoboCup 4-Legged League (that used Sony AIBOs up till 2008), that was later renamed the RoboCup Standard Platform League (that uses Aldebaran NAOs from 2008), every team in the league is required to use the same robot hardware. Thus, the focus of the league is to develop algorithms without having to optimize the hardware.

  However, our experience at RoboCup is that "homogeneous" robots are not always homogeneous. In particular, robots may have slight manufacturing differences (especially the NAOs that are hand-assembled by engineers), and varying wear-and-tear. As a result, different robots have slightly different capabilities. For example, one robot may be able to walk more quickly and with less odometry error than another, or a robot may be able to perform multiple types of kick stably while other robots are only able to perform a subset of the kicks due to wear-and-tear on the joints.

  Our strategy for selecting the initial roles of the robots for the RoboCup games was to specifically select the main attacker robot and goalkeeper. Typically, the robot with the fastest walking and most stable kicks would be the attacker, and the robot that had a poor walk would be the goalkeeper (since the goalkeeper seldom moves). Although the role assignment algorithm would dynamically assign the attacker and supporter roles to the field players during the game, the main attacker that we picked would always be positioned closest to the ball at the beginning of the game and after goals were scored, so it would run the attacker behavior most of the time.

  From these experiences, I sought to develop an algorithm that would quantitatively learn the robots' capabilities and perform the team selection that we had been manually doing. The Synergy Graph learning and team formation algorithms were a first step to solving the problem, and future work would improve upon these algorithms to apply them to scenarios such as RoboCup.

- **Taking risks can have huge payoffs**

  In previous years, our RoboCup behaviors tended to be "cautious", where a robot would only kick the ball when it was sure of the opposing goal's location. Such a strategy prevents the ball from needlessly being kicked out of bounds, which causes the ball to be replaced behind the robot and is a negative consequence. However, the robots tend to take extra time to confirm their position before kicking, which gives time for the opposing team to approach the ball and block the shot to goal.

  Recently, we have experimented with taking "risky" shots at goal. One method is for the robot to kick a ball as long as it will travel up-field — a ball going up-field may not necessarily enter the opposing goal, but makes it easier for a subsequent shot to enter. A second method is to kick the ball towards the goal even if the localization estimate of the robot's position is imprecise. The overall idea of taking such risks is to maximize the number of shots on the goal. Instead of taking a small number of shots on goal with a high probability of success, we opted to take more shots on goal with a lower probability of success, so that overall more goals will be scored.

  The variability of team performance at RoboCup was a strong motivation for me to consider random variables to model capabilities and not utility values. Further, the trade-off between risk and reward prompted me to pursue the concept of δ-optimality in this thesis. An interesting future direction would be to consider how to pick δ so as to optimize the end result of the task.

- **Every team is developed independently**

  Currently, each team that participates in RoboCup develops its algorithms independently. Some teams release their source code and documentation after the competition is completed, which other teams may then use in the future competitions. However, for any given year, every team has its own set of algorithms to handle team coordination, role switching, communication, etc.

  There have been efforts to create an ad hoc robot soccer team (also known as a pickup soccer team), where different teams provide different robots who play on the same team. Such an endeavor requires either some form on implicit communication, e.g., a robot does not approach the ball when it sees a teammate near the ball, or explicit communication with some pre-defined protocols.

  The concept of ad hoc robot soccer appealed to me, and while I am interested in how to design an algorithm that can play soccer with new teammates, I am primarily interested in how a "coach" of such a team would function. The goal of the coach would be to select

the best robots for the team, and possibly also "train" them to perform better. This concept of a "coach" has been a primary motivation for the thesis to consider ad hoc teams, and also learning agents. Future work would include more development of algorithms for the coach, such as how observations of the team performance are made, and how to improve the robots' behaviors during the task.

9.3 Future Directions

The Synergy Graph model is general and applicable to many interesting multi-robot problems. We enumerate a number of directions for future work that are closely related to the scientific contributions of this thesis:

- **Incorporating details of team performance**
  In this thesis, we treated team performance as a value returned by a black-box, and the Synergy Graph model used these values to learn its structure and agent capabilities. For example, when we performed the real robot experiments, we hand-crafted functions that computed the team performance using features such as the amount of time the team took to complete the task, the number of items foraged, etc. It would be interesting if the performance function was revealed to the Synergy Graph, so that it may learn the underlying features of the team and predict its performance more accurately.

- **Fully-iterative learning and team formation**
  The approach of this thesis consists of two discrete phases: learning the Synergy Graph from observations, and then using the learned Synergy Graph to form a team. An interesting approach would be to learn an initial Synergy Graph from a limited number of observations, and use an active-learning approach to form teams and obtain new observations in order to improve the learned Synergy Graph.

- **Modeling beyond pairwise relationships**
  The Synergy Graph model uses pairwise relationships as the building block for computing synergy. An extension to the model would be to consider n-ary relationships, e.g., three specific agents that have high performance as a team, but have low performance individually or in pairs.

- **Modeling non-Gaussian capabilities**
  The Synergy Graph model uses Gaussian variables to represent the capabilities of the

agents, and Gaussian mixture models to represent team performance when failures may occur. An interesting direction would be to consider other (possibly non-parametric) distributions of capabilities and performance.

- **Learning the Synergy Graph using expectation-maximization**
  The Synergy Graph learning algorithm presented in this thesis iteratively improved the Synergy Graph structure, and learned the agent capabilities given some structure. A future direction would be to consider the expectation-maximization algorithm to improve the structure and agent capabilities, which may provide faster convergence and/or a better learned result.

- **Proving bounds to the learning and team formation algorithms**
  The learning algorithms and team formation algorithms presented in this thesis were empirically evaluated and shown to be effective. However, it would be useful to prove bounds on the algorithms such that there is a measure of the worst case result of the algorithms, especially when Synergy Graphs are applied to real problems.

- **Selecting δ to maximize performance**
  This thesis assumed that the probability δ is given, and finds the δ-optimal team. It would be interesting to consider how δ can be selected so as to maximize some other optimality criterion, for example to "hedge bets" on performance.

- **Applying Synergy Graphs to human teams**
  The experiments in this thesis considered multi-robot and multi-agent teams. An interesting future direction would be to consider human teams, and whether the Synergy Graph model sufficiently captures relationships among humans. Some possible domains include sports (such as basketball and baseball where detailed information about players and games are available) and business (such as organizational teams commonly considered in operations research). Further, humans are able to learn through experience, and their performance may not be readily observable, so it would be interesting to consider how to model and learn Synergy Graphs for human teams.

9.4 Concluding Remarks

This thesis has shown that the Synergy Graph model is general and applicable to many multi-agent and multi-robot problems; the learning and team formation algorithms only require observations of team performance and no other domain information. The Synergy Graph model is easily extended to capture many interesting characteristics, such as those detailed in this thesis. Extensive empirical evaluation of the Synergy Graph model has been performed using simulated and real robots, that demonstrate that teams formed by applying the Synergy Graph are effective at solving the task and obtaining high performance. In this thesis, we have focused our experiments on teams of 2-3 robots, and we believe that the Synergy Graph model is applicable to larger teams. At the same time, we believe that the notion of synergy may not be applicable to huge teams (e.g., more than 50 robots) since such large teams will likely be decomposable into smaller sub-teams performing independent sub-tasks; the Synergy Graph would then be applicable to model the synergy of the smaller sub-teams.

Bibliography

[Agmon and Stone, 2012] Agmon, N. and Stone, P. (2012). Leading Ad Hoc Agents in Joint Action Settings with Multiple Teammates. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 341–348. 8.4

[Agustin-Blas et al., 2011] Agustin-Blas, L., Salcedo-Sanz, S., Ortiz-Garcia, E., Portilla-Figueras, A., Perez-Bellido, A., and Jimenez-Fernandez, S. (2011). Team formation based on group technology: A hybrid grouping genetic algorithm approach. *Journal of Computers & Operations Research*, 38(2):484–495. 8.6

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3):235–256. 6.2.2

[Bachrach et al., 2010] Bachrach, Y., Meir, R., Jung, K., and Kohli, P. (2010). Coalitional Structure Generation in Skill Games. In *Proceedings of the International Conference on Artificial Intelligence*. 8.3

[Banerjee and Kraemer, 2010] Banerjee, B. and Kraemer, L. (2010). Coalition Structure Generation in Multi-Agent Systems with Mixed Externalities. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 175–182. 8.3

[Barrett and Stone, 2011] Barrett, S. and Stone, P. (2011). Ad Hoc Teamwork Modeled with Multi-armed Bandits: An Extension to Discounted Infinite Rewards. In *Proc. Int. Conf. Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop*. 8.4

[Barrett et al., 2011] Barrett, S., Stone, P., and Kraus, S. (2011). Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 567–574. 6.2.1, 8.4

[Barrett et al., 2012] Barrett, S., Stone, P., Kraus, S., and Rosenfeld, A. (2012). Learning Teammate Models for Ad Hoc Teamwork. In *Proc. Int. Conf. Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop*. 8.4

[Boon and Sierksma, 2003] Boon, B. and Sierksma, G. (2003). Team formation: Match-

ing Quality Supply and Quality Demand. *European Journal of Operational Research*, 148(2):277–292. 8.5, 8.6

[Bowling et al., 2003] Bowling, M., Browning, B., Chang, A., and Veloso, M. (2003). Plays as Team Plans for Coordination and Adaptation. In *Proceedings of the RoboCup International Symposium*, pages 686–693. 8.2

[Browning et al., 2005] Browning, B., Bruce, J., Bowling, M., and Veloso, M. (2005). STP: Skills, Tactics and Plays for Multi-Robot Control in Adversarial Environments. *IEEE Journal of Systems and Controls Engineering*, 219(1). 8.2

[Bulka et al., 2007] Bulka, B., Gaston, M., and desJardins, M. (2007). Local Strategy Learning in Networked Multi-Agent Team Formation. *Journal of Autonomous Agents and Multi-Agent Systems*, 15:29–45. 8.5

[Chakraborty and Stone, 2013] Chakraborty, D. and Stone, P. (2013). Cooperating with a Markovian Ad Hoc Teammate. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1085–1092. 8.4

[Chalkiadakis and Routilier, 2010] Chalkiadakis, G. and Routilier, C. (2010). Sequential Optimal Repeated Coalition Formation under Uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 1–44. 8.3

[Chen and Sun, 2011] Chen, J. and Sun, D. (2011). Resource Constrained Multirobot Task Allocation Based on Leader-Follower Coalition Methodology. *Journal of Robotics Research*, 30(12):1423–1434. 8.1

[Chen and Wang, 2007] Chen, Y. and Wang, Y. (2007). Obstacle Avoidance and Role Assignment Algorithms for Robot Formation Control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4201–4206. 8.2

[Coltin et al., 2010] Coltin, B., Liemhetcharat, S., Ç. Meriçli, Tay, J., and Veloso, M. (2010). Multi-Humanoid World Modeling in Standard Platform Robot Soccer. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*. 8.2

[Cunningham et al., 2012] Cunningham, A., Wurm, K., Burgard, W., and Dellaert, F. (2012). Fully distributed scalable smoothing and mapping with robust multi-robot data association. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1093–1100. 8.7

[Dastani et al., 2003] Dastani, M., Dignum, V., and Dignum, F. (2003). Role-Assignment in Open Agent Societies. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 489–496. 8.2

[de Weerdt et al., 2007] de Weerdt, M., Zhang, Y., and Klos, T. (2007). Distributed Task Allocation in Social Networks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 500–507. 8.1, 8.5

[Dereli et al., 2007] Dereli, T., Baykasoglu, A., and Das, G. (2007). Fuzzy quality-team formation for value added auditing: A case study. *Journal of Engineering Technology Management*, 24(4):366–394. 8.6

[Dias, 2004] Dias, M. B. (2004). *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, The Robotics Institute, Carnegie Mellon University. 8.1

[Dias and Stentz, 2000] Dias, M. B. and Stentz, A. (2000). A Free Market Architecture for Distributed Control of a Multirobot System. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 115–122. 8.1

[Dias and Stentz, 2002] Dias, M. B. and Stentz, A. (2002). Multi-Robot Exploration Controlled By A Market Economy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2714–2720. 8.1

[Dobson and Goddard, 2010] Dobson, S. and Goddard, J. (2010). Optimizing Strategic Behaviour in a Dynamic Setting in Professional Team Sports. *European Journal of Operational Research*, 205(3):661–669. 8.6

[Dorn and Dustdar, 2010] Dorn, C. and Dustdar, S. (2010). Composing Near-Optimal Expert Teams: A Trade-off between Skills and Connectivity. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 472–489. 2.2, 8.5

[dos Santos and Bazzan, 2011] dos Santos, F. and Bazzan, A. L. C. (2011). Towards Efficient Multiagent Task Allocation in the RoboCup Rescue: A Biologically-Inspired Approach. *Journal of Autonomous Agents and Multi-Agent Systems*, 22:465–486. 7.2.1

[Feng et al., 2010] Feng, B., Jiang, Z., Fan, Z., and Fu, N. (2010). A Method for Member Selection of Cross-Functional Teams using the Individual and Collaborative Performances. *European Journal of Operational Research*, 203(3):652–661. 8.6

[Ferreira et al., 2010] Ferreira, P., Santos, F. D., Bazzan, A. L., Epstein, D., and Waskow, S. J. (2010). RoboCup Rescue as Multiagent Task Allocation among Teams: Experiments with Task Interdependencies. *Journal of Autonomous Agents and Multi-Agent Systems*, 20:421–443. 7.2.1

[Frias-Martinez et al., 2004] Frias-Martinez, V., Sklar, E., and Parsons, S. (2004). Exploring Auction Mechanisms for Role Assignment in Teams of Autonomous Robots. In *Proceedings*

of the RoboCup International Symposium, pages 532–539. 8.2

[Gaston and desJardins, 2005] Gaston, M. and desJardins, M. (2005). Agent-Organized Networks for Dynamic Team Formation. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 230–237. 8.5

[Gaston et al., 2004] Gaston, M., Simmons, J., and desJardins, M. (2004). Adapting Network Structure for Efficient Team Formation. In *Proceedings of the AAAI Symposium on Articial Multi-agent Learning*. 8.5

[Genter et al., 2011] Genter, K., Agmon, N., and Stone, P. (2011). Role-Based Ad Hoc Teamwork. In *Proceedings of the Plan, Activity, and Intent Recognition Workshop at the Twenty-Fifth Conference on Artificial Intelligence (PAIR-11)*. 8.4

[Genter et al., 2013] Genter, K., Agmon, N., and Stone, P. (2013). Ad Hoc Teamwork for Leading a Flock. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 531–538. 8.4

[George et al., 2010] George, J. M., Pinto, J., Sujit, P. B., and Sousa, J. B. (2010). Multiple UAV Coalition Formation Strategies (Extended Abstract). In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1503–1504. 8.3

[Gerkey and Mataric, 2004] Gerkey, B. P. and Mataric, M. J. (2004). A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *Journal of Robotics Research*, 23(9):939–954. 1, 7.2.1, 8.1, 8.1, 8.1

[Guttmann, 2008] Guttmann, C. (2008). Making allocations collectively: Iterative group decision making under uncertainty. In *Proceedings of the 6th German Conference on Multiagent System Technologies*, pages 73–85. 8.1

[He and Ioerger, 2003] He, L. and Ioerger, T. R. (2003). A quantitative model of capabilities in multi-agent systems. In *Proceedings of the International Conference on Artificial Intelligence*, pages 730–736. 8.1

[Hirotsu and Wright, 2003] Hirotsu, N. and Wright, M. (2003). Determining the Best Strategy for Changing the Configuration of a Football Team. *Journal of the Operational Research Society*, 54(8):878–887. 8.6

[Isik et al., 2006] Isik, M., Stulp, F., Mayer, G., and Utz, H. (2006). Coordination without negotiation in teams of heterogeneous robots. In *Proceedings of the RoboCup International Symposium*, pages 355–362. 8.4

[Ji et al., 2006] Ji, M., Azuma, S., and Egerstedt, M. (2006). Role assignment in multi-agent coordination. *International Journal of Assistive Robotics and Mechatronics*, 7(1):32–40. 8.2

[Jones et al., 2006] Jones, E. G., Browning, B., Dias, M. B., Argall, B., Veloso, M., and Stentz, A. (2006). Dynamically Formed Heterogeneous Robot Teams Performing Tightly-Coordinated Tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 570–575. 8.2

[Kaminka and Tambe, 2000] Kaminka, G. and Tambe, M. (2000). Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147. 8.7

[Kitano et al., 1999] Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 739–743. 7.2.1

[Lappas et al., 2009] Lappas, T., Liu, K., and Terzi, E. (2009). Finding a Team of Experts in Social Networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 467–476. 2.2, 8.5

[Le et al., 2011] Le, V., Stinckwich, S., Noury, B., and Doniec, A. (2011). Dynamic role assignment for large-scale multi-agent robotic systems. In Bai, Q. and Fukuta, N., editors, *Advances in Practical Multi-Agent Systems*, volume 325 of *Studies in Computational Intelligence*, pages 311–326. Springer Berlin / Heidelberg. 8.2

[Li and Shan, 2010] Li, C. and Shan, M. (2010). Team Formation for Generalized Tasks in Expertise Social Networks. In *Proceedings of the International Conference on Social Computing*, pages 9–16. 8.5

[Liemhetcharat et al., 2010] Liemhetcharat, S., Coltin, B., and Veloso, M. (2010). Vision-Based Cognition of a Humanoid Robot in Standard Platform Robot Soccer. In *Proceedings of the International Workshop on Humanoid Soccer Robots*, pages 47–52. 8.2

[Liemhetcharat and Veloso, 2011] Liemhetcharat, S. and Veloso, M. (2011). Modeling Mututal Capabilities in Heterogeneous Teams for Role Assignment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3638–3644. 2.7

[Liemhetcharat and Veloso, 2012a] Liemhetcharat, S. and Veloso, M. (2012a). Modeling and Learning Synergy for Team Formation with Heterogeneous Agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 365–375. Nominated for Best Student Paper Award. 2, 3, 7

[Liemhetcharat and Veloso, 2012b] Liemhetcharat, S. and Veloso, M. (2012b). Weighted Synergy Graphs for Role Assignment in Ad Hoc Heterogeneous Robot Teams. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5247–5254.

5, 7

[Liemhetcharat and Veloso, 2013a] Liemhetcharat, S. and Veloso, M. (2013a). Forming an Effective Multi-Robot Team Robust to Failures. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 6, 7

[Liemhetcharat and Veloso, 2013b] Liemhetcharat, S. and Veloso, M. (2013b). Learning the Synergy of a New Teammate. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4

[Liemhetcharat and Veloso, 2013c] Liemhetcharat, S. and Veloso, M. (2013c). Synergy Graphs for Configuring Robot Team Members. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 111–118. 5, 7

[Liemhetcharat and Veloso, 2013d] Liemhetcharat, S. and Veloso, M. (2013d). Weighted Synergy Graphs for Effective Team Formation with Heterogeneous Ad Hoc Agents. *Journal of Artificial Intelligence*. Under Revision. 2, 3, 7

[Lim et al., 2009] Lim, C., Mamat, R., and Braunl, T. (2009). Market-based Approach for Multi-Team Robot Cooperation. In *Proceedings of the International Conference on Autonomous Robots and Agents*, pages 62–67. 8.1

[McMillen and Veloso, 2006] McMillen, C. and Veloso, M. (2006). Distributed, Play-Based Role Assignment for Robot Teams in Dynamic Environments. In *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems*, pages 145–154. 8.2

[Michalak et al., 2010] Michalak, T., Marciniak, D., Szamotulski, M., Rahwan, T., Wooldridge, M., McBurney, P., and Jennings, N. (2010). A Logic-Based Representation for Coalitional Games with Externalities. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 125–132. 8.3

[Napp and Klavins, 2010] Napp, N. and Klavins, E. (2010). Robust by composition: Programs for multi-robot systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2459–2466. 8.7

[Parker, 1998] Parker, L. (1998). ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions of Robotics and Automation*, 14(2):220–240. 8.7

[Parker, 2011] Parker, L. (2011). Reliability and Fault Tolerance in Collective Robot Systems. In Kernbach, S., editor, *Handbook on Collective Robotics*. Pan Stanford Publishing. 8.7

[Parker and Tang, 2006] Parker, L. and Tang, F. (2006). Building Multirobot Coalitions Through Automated Task Solution Synthesis. *Proceedings of the IEEE*, 94(7):1289–1305. 7, 7.1, 7.1.2, 7.7, 8.1, 8.5

[Preisler and Renz, 2012] Preisler, T. and Renz, W. (2012). Scalability and robustness analysis of a multi-agent based self-healing resource-flow system. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 1216–1268. 8.7

[Rahwan et al., 2009] Rahwan, T., Michalak, T., Jennings, N., Wooldridge, M., and McBurney, P. (2009). Coalition Formation with Spatial and Temporal Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 257–263. 8.3

[Ramchurn et al., 2010] Ramchurn, S., Polukarov, M., Farinelli, A., and Truong, C. (2010). Coalition Formation with Spatial and Temporal Constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1188. 7.2.1, 8.3

[RoboCupRescue, 2011] RoboCupRescue (2011). RoboCup Rescue Simulation League. `http://roborescue.sourceforge.net/`. [Online]. 7.2.2, 7.7

[RoboCupSPL, 2013] RoboCupSPL (2013). RoboCup Standard Platform League. `http://www.tzi.de/spl/`. [Online]. 1.1.4

[Roth et al., 2003] Roth, M., Vail, D., and Veloso, M. (2003). A Real-time World Model for Multi-Robot Teams with High-Latency Communication. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2494–2499. 8.2

[Sandholm et al., 1999] Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohme, F. (1999). Coalition Structure Generation with Worst Case Guarantees. *Journal of Artificial Intelligence*, 111:209–238. 1, 8.3, 8.3

[Sen and Adams, 2013] Sen, S. and Adams, J. (2013). A Decision Network based Framework for Multiagent Coalition Formation. In *Proc. Int. Conf. Autonomous Agents Multiagent Systems*, pages 55–62. 8.3

[Service and Adams, 2011a] Service, T. and Adams, J. (2011a). Coalition formation for task allocation: theory and algorithms. *Journal of Autonomous Agents and Multi-Agent Systems*, 22:225–248. 8.1, 8.3, 8.3

[Service and Adams, 2011b] Service, T. and Adams, J. (2011b). Constant factor approximation algorithms for coalition structure generation. *Journal of Autonomous Agents and Multi-Agent Systems*, 23:1–17. 8.3

[Shehory and Kraus, 1998] Shehory, O. and Kraus, S. (1998). Task Allocation via Agent Coalition Formation. *Journal of Artificial Intelligence*, 101(1-2):165–200. 1, 8.1

[Simmons et al., 2000] Simmons, R., Singh, S., Hershberger, D., Ramos, J., and Smith, T. (2000). First results in the coordination of heterogeneous robots for large-scale assembly.

In *Proceedings of the International Symposium of Experimental Robotics*, pages 323–332. 8.2

[Stein et al., 2011] Stein, S., Gerding, E., Rogers, A., Larson, K., and Jennings, N. (2011). Algorithms and Mechanisms for Procuring Services with Uncertain Durations using Redundancy. *Journal of Artificial Intelligence*, 175(14-15). 8.7

[Stone et al., 2010] Stone, P., Kaminka, G., Kraus, S., and Rosenschein, J. (2010). Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *Proceedings of the International Conference on Artificial Intelligence*. 1, 1.1.1, 6, 6.2.1, 8.4

[Stone et al., 2009] Stone, P., Kaminka, G., and Rosenschein, J. (2009). Leading a Best-Response Teammate in an Ad Hoc Team. In *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 132–146. 8.4

[Stone and Kraus, 2010] Stone, P. and Kraus, S. (2010). To Teach or not to Teach? Decision Making Under Uncertainty in Ad Hoc Teams. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 117–124. 8.4

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork. *Journal of Artificial Intelligence*, 110(2):241–273. 8.2

[Tang and Parker, 2007] Tang, F. and Parker, L. (2007). A Complete Methodology for Generating Multi-Robot Task Solutions using ASyMTRe-D and Market-Based Task Allocation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3351–3358. 8.1, 8.1, 8.5

[Thompson, 1933] Thompson, W. (1933). On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294. 6.2.2

[Tosic and Agha, 2004] Tosic, P. and Agha, G. (2004). Maximal Clique Based Distributed Coalition Formation for Task Allocation in Large-Scale Multi-Agent Systems. In *Proceedings of the International Workshop on Massively Multi-Agent Systems*, pages 104–120. 8.1

[Vail and Veloso, 2003] Vail, D. and Veloso, M. (2003). Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume II*, pages 87–100. 8.2

[Vig and Adams, 2006a] Vig, L. and Adams, J. (2006a). Market-based Multi-Robot Coalition Formation. In *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems*, pages 227–236. 8.1

[Vig and Adams, 2007] Vig, L. and Adams, J. (2007). Coalition Formation: From Software

Agents to Robots. *Journal of Intelligent Robotic Systems*, 50:85–118. 8.1, 8.3, 8.3

[Vig and Adams, 2006b] Vig, L. and Adams, J. A. (2006b). Multi-Robot Coalition Formation. *IEEE Transactions of Robotics*, 22(4):637–649. 8.3

[Wu et al., 2011] Wu, F., Zilberstein, S., and Chen, X. (2011). Online Planning for Ad Hoc Autonomous Agent Teams. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 439–445. 8.4

[Zakarian and Kusiak, 1999] Zakarian, A. and Kusiak, A. (1999). Forming teams: an analytical approach. *IIE Transactions*, 31(1):85–97. 8.6

[Zhang and Parker, 2010] Zhang, Y. and Parker, L. (2010). IQ-ASyMTRe: Synthesizing Coalition Formation and Execution for Tightly-Coupled Multirobot Tasks. In *Proc. IEEE Int. Conf. Intelligent Robots Systems*, pages 5595–5602. 7.2.2, 8.1

[Zhang and Parker, 2012] Zhang, Y. and Parker, L. (2012). Task allocation with executable coalitions in multirobot tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 1, 7.2.2, 8.1

[Zhang and Parker, 2013] Zhang, Y. and Parker, L. (2013). Considering Inter-Task Resource Constraints in Task Allocation. *Journal of Autonomous Agents and Multi-Agent Systems*, 26(3):389–419. 8.1

[Zhu and Alkins, 2009] Zhu, H. and Alkins, R. (2009). Group Role Assignment. In *Proceedings of the International Symposium on Collaborative Technologies and Systems*, pages 431–439. 8.2

[Zlot et al., 2002] Zlot, R., Stentz, A., Dias, M. B., and Thayer, S. (2002). Multi-Robot Exploration Controlled By A Market Economy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3016–3023. 8.1