A Study of Dynamic Coordination Mechanisms

Avi Rosenfeld
Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University
Ramat-Gan, Israel
February 2007

This work was carried out under the supervision of

Dr. Gal A. Kaminka and Prof. Sarit Kraus
Department of Computer Science
Bar-Ilan University

This work is dedicated to my loving wife, Esther.

Acknowledgments

In Hebrew there is a blessing typically said around holidays and special occasions, "shehecheyanu vekiyimanu vehigianu lazman hazeh", or thanking God for allowing us to reach certain events within our lives. Whether or not this blessing should be said when one submits their PhD dissertation is an interesting and technical question. Nonetheless, I believe submitting this thesis is quite a milestone in my life, and requires many acknowledgements of thanks.

First, and foremost, I must thank God. I have considered myself very fortunate in life, and have been blessed by many supportive family members and friends. Foremost, I give thanks to God for guiding my decisions and bringing my path in contact with such supportive people.

Almost six years ago (nearly to this day) I decided to move to Israel, and pursue a PhD. At the time, I knew the direction I wanted to take, but did not truly understand the magnitude of this undertaking. There were many challenges along way: a new country with a new language and different cultural norms, loss of family support, long hours spent with challenging courses and research, just to name a few. Despite all of these challenges, I must admit that I do not regret the decision to move to Israel or to pursue my PhD at Bar-Ilan.

My advisors, Gal Kaminka and Sarit Kraus were of invaluable help and support. I am grateful for having two wonderful mentors who guided me. Gal was always full of enthusiasm and interesting ideas. He was the one who recommended first pursuing the robotic domain within the first two chapters of this dissertation. In addition to his academic support, he was extremely helpful in smoothing out some of the personal bumps along way. I still remember some of his advice from our first conversation about moving to Israel. Sarit is one of the most extraordinary people I have ever met. She seems to have an infinite amount of time for her family and students. I am amazed by the amount she is able to accomplish. I would also like to thank her for connecting me with NSF and DARPA funding which was my primary support during this process. Without this financial support, I would not have been able to complete this program. I am truly indebted to both of you.

I would also like to acknowledge the other funding I have received throughout my PhD. First, the Computer Science Department of Bar-Ilan was extremely generous, and provided me funding and an exemption from tuition from my first semester. Starting in my second year, I received a fellowship from Israel's Absorb-

tion department for funding new Israeli (*Olim*) PhD students. In my last full year I received the Schupf fellowship, which generously supports PhD students with leadership potential. In addition to the sponsors of these grants, I am grateful to all of the administrators who helped insure that my funding was transferred in a timely fashion.

I would also like to acknowledge all of my collaborators. First, the Maverick and DAI labs at Bar-Ilan were always my academic home. All of the lab members are talented researchers who stimulated many fruitful discussions. I would especially like to mention Noa Segel-Argamon, Michal Chalamish, Ariel Felner, Meirav Hadad, Meir Kalech, Efrat Manister, Dudi Sarne, Osher Yadgar and Aner Yarden for their help throughout my time at Bar-Ilan. I would like to single out Noa who was extremely helpful in formulating several concepts in Chapters 2 and 3.

The NSF and DARPA projects I worked on introduced me to many interesting people outside of Bar-Ilan. I am indebted to Barbara Grosz of Harvard who spent many hours giving encouragement, and stimulating many interesting conversations. Willem-Jan van Hoeve of Cornell developed the scheduler used in Chapter 4 of this thesis. Charlie Ortiz, Regis Vincent, and Roger Mailler of SRI stimulated several interesting conversations of this chapter well. I consider myself fortunate for having the opportunity to work with these people and their groups.

Finally, my family has been a source of strength and support throughout this process. I thank my children, Eliyahu, Hadassah and Shelomoh for their understanding of the long hours that have accompanied this process. Netanel arrived fairly late into this process (only two months ago). I would like to thank all of them for making sure I never needed to worry about oversleeping. My parents and grandparents were always very supportive of me. I am sure my father-in-law will be happy there will be another PhD in the family! Finally, I am deeply in gratitude to my wife, Esther. She has been extremely loving and supportive throughout this entire process, at many times, at her own expense. Time after time, she coached me through the ups and downs of this process. This PhD could never have happened without her. It is for this reason I am dedicating this work to her.

Contents

List of Algorithms

List of Figures

List of Tables

Abstract

Coordination, or the act of managing interdependencies between activities, is a key issue within the field of multi-agent systems. Because of the importance of this issue, many theoretical and practical frameworks have been proposed for addressing coordination challenges. However, finding the optimal coordination method for a given group of agents and domain task is a computationally difficult, if not intractable, problem in most real-world domains. Solving the "coordination problem" is thus an important open challenge for researchers in this field.

Towards addressing this issue, this thesis presents an algorithm selection approach for creating adaptive coordination methods. We study several types of coordination problems from robotic foraging and search domains, constraint satisfaction and optimization domains, and Peer to Peer networks. We find that novel teamwork measures can be developed for quantifying the effectiveness of coordination algorithms in all of these domains. These measures can be autonomously and locally measured by team members, even without any communication. The significance of this result is its ability to effectively quantify coordination in a clear, tractable fashion. Next, we find that these measures can be used to switch between coordination methods as needed. Robots or agents can effectively select the best coordination method to their localized domain conditions, online during task execution. The net result is a significant productivity improvement of these adaptive methods over the static methods they are based on.

Chapter 1

Introduction

Coordination can be defined as "managing dependencies between activities" [49]. Effective coordination is required to manage the dependencies that naturally occur when agents have inter-linked objectives or share a common environment or resources in a multi-agent system (MAS) [17]. Coordination is a cross-discipline topic researched within the fields of organization theory, psychology, economics, and artificial intelligence [49].

However, while there are a number of theoretical definitions for coordination across these fields, generally finding optimal or near optimal coordination techniques remains an open question. Within artificial intelligence, several coordination frameworks have been suggested to date for reasoning about teamwork and coordination [25, 39, 69]. For example, one approach is to use decision theoretic models such as Markov Decision Processes (MDP) [60] within any of these formalized frameworks. This could potentially allow robots to choose the optimal coordination method as needed during task completion.

However, while each of these approaches has been shown to be effective under certain conditions, in many real-world applications the problem of making the optimal coordination decision is computationally intractable [60]. The inherent complexity in using these

approaches demonstrates the necessity of creating novel approaches to effectively deal with real-world coordination issues in a tractable fashion.

Towards addressing this issue, this thesis presents an algorithm selection approach for creating adaptive coordination methods. Similar to previous work [17], we assume an agent has a library of coordination algorithms at its disposal, with each algorithm having different attributes. These differences make each algorithm best suited to different types of tasks and environments. As a result, the key question is how can one select the best algorithm given a specific coordination problem instance at hand.

The first step towards addressing this challenge is to develop novel teamwork measures to quantify the relative effectiveness of coordination methods. Once these measures have been identified, we can identify what is the best coordination algorithm given a specific interaction between agents. This allows us to tractably analyze which coordination methods to use. This information can then be applied to create effective dynamic coordination heuristics that selects the best coordination method. This approach has several key advantages:

- **Quick Analysis of Coordination** - Differences between algorithms are quickly identified during very short learning periods. This allows us to create adaptive coordination policies without exhaustive learning in a state space that is often exponentially large.

- **Online adaptation** - The coordination adaptation policies created in this approach are suitable for online use. There is no need to preplan or preprocess all possible group interactions.

- **Significant Productivity Gains** - We find that this approach

consistently achieved a significant improvement in productivity over the static methods they were based on in a variety of real-world domains.

This thesis describes how this approach was applied to several diverse coordination domains including: robotic groups with and without communication, optimization domains such as scheduling and graph coloring, and full text search within Peer to Peer networks. In all cases, we were able to identify key attributes that differentiated between that domain's coordination algorithms. Using this information we then successfully created dynamic coordination methods that significantly improved the group's performance.

1.1 Background

The contributions in this thesis are related to a variety of existing research topics including: teamwork models [25, 39, 49, 69], teamwork measures [7, 26, 35], algorithm selection [2, 23, 38, 40], and dynamic coordination [17]. To understand the significance of the contribution of this thesis, we first present related work from these areas.

1.1.1 Teamwork Models

Because of the importance of coordination problems, a variety of teamwork frameworks and formalizations have been proposed within the multi-agent research community [39, 25, 69]. The SharedPlans approach [25] consists of creating teamwork Recipes based on modeling agents' beliefs and intentions. Jenning's joint intentions model [31] involves flexible reasoning about joint commitments and their associated social conventions. Tambe's STEAM teamwork engine [69] provides a

generalized teamwork engine. The TAEMS framework [39] consists of a rule based approach to quantifying coordination relationships.

However, within each of these frameworks, individual agents must reason about the next action it will take, and how this action will impact the group. One popular formalization is to modeling this coordination action problem as a multi-agent Markov Decision Process (MMDP). This process was previously modeled as follows [60]:

Every agents' actions can be described as a tuple, <S,$A_{i=1}^{N}$, Pr, R> where:

- N is a group of agents engaged in some cooperative behavior.
- S is the set of all possible environment states for these agents.
- A_i is the set of all possible joint actions applicable in the environment by agent i.
- Pr models the system's dynamics. Pr takes the form: S $\times$ A $\times$ S $\rightarrow$ [0,1]; with Pr(s_i, a, s_j) denoting the probability that action a executed in state s_i, will transition to state s_j.
- R is the reward function for each agent's possible action.

The advantage to this model is in its clarity and ability to describe optimal behavior. Formally, for any time horizon T, a policy π can be created such that π: S $\times$ (1, . . ., T) $\rightarrow$ A is associated with every state s, and for all future time steps, t, such that t $\leq$ T. An optimal policy requires that for every action, a, taken in A, the reward function R(a) $\geq$ R(a$'$) for all a$' \epsilon A$.

However, practically using this model to find an optimal policy is an intractable problem in all but the most basic domains for the following reasons [60]:

- Even the basic MDP model requires the system designer to quantify the state values, S, of all possible actions, A_i, and the probability functions Pr associated with every agent's possible action. Many real-world domains cannot be quantified in this fashion.

- In MMDP models, the designer must additionally quantify the impact other agent's actions will have on its own actions – another difficult factor to model, often resulting in an exponential jump in the state space size. Finding an optimal solution in many of these problems is NEXP-complete – a class of one of the hardest problems sets currently known to computer scientists.

- Both MDP and MMDP models assume agents can accurately quantify its current state. Once this assumption fails, a Partially Observable model (PoMDP) must be used, further increasing the complexity of the problem.

This intractability is one of the motivations for our approach.

The first contribution of this thesis is in identifying and developing novel teamwork measures that can be used to quantify interactions between agents. To date there have been very few studies contrasting a group's composition and its task performance. Balch [7] presented a metric of *social entropy* which can measure the level of diversity or how heterogeneous a group is. He claimed that certain tasks are intrinsically better suited for homogeneous groups, with others for heterogeneous ones. He found his measure positively correlated with the group's productivity in some domains, and negatively correlated in others. Kaminka and Tambe [35] used an *average time to agreement* (ATA) measure to study a team's behavior in Robocup and ModSAF agent domains. This measure was used to evaluate the relative effectiveness of social monitoring of team behaviors. Similar to this work,

the ATA measure aims to provide feedback about team effectiveness. However, again the question of correlation between productivity and the ATA measure was left open.

Hogg and Jennings [26] introduce a *willingness to cooperate factor* which defines the degree to which a social agents engage in individual versus group considerations. Similar to the approach we present, they use their measure to alter agents' activity to resource constraints that are sensed during run-time. However, their formalized structure is less flexible to change than ours and requires a Q-learning model to allow for adaptation. As a result, they left for future work how their model could be applied for quickly reacting to domain dynamics, or how agents can manage activities between different subgroups they might be a part of. These are issues that critical to the coordination problem we study.

1.1.2 Algorithm Selection

Once we have demonstrated that we can quantify the relative effectiveness of coordination methods, this information is used to select the best method for a given group composition and domain. This can be viewed as a contribution within the field of algorithm selection. Previously, Rice defined this problem as the process of choosing the best algorithm for any given instance of a problem from a given set of potential algorithms [63].

Following this approach, we define the coordination algorithm selection process used in this thesis as follows: Let $G = \{a_1, \ldots, a_N\}$ be a group on N agents engaged in some cooperative behavior. Each agent, a, can choose out of a library of coordination algorithms, $\{CA_1 \ldots CA_k\}$. We denote this selection CA_{a_j}, where $1 \leq a_j \leq k$.

Using CA_{a_j} affects the group's utility by a certain value, $\mathcal{UT}^a(CA_{a_j})$. $\mathcal{UT}^a(CA_{a_j})$ is composed of a gain, $\mathcal{G}(CA_{a_j})$, that the group will realize by using algorithm CA_{a_j}, and the agent's cost, $\mathcal{C}(CA_{a_j})$, by using that same algorithm. As this thesis deals with groups of cooperative agents, the goal is to maximize $\sum_{a=1}^{N} \mathcal{UT}^a(CA_{a_j})$. To achieve this, each agent must select the algorithm CA_{a_j} from the k possible algorithms in the library whose value $\mathcal{G}(CA_{a_j})$ - $\mathcal{C}(CA_{a_j})$ is highest.

The correct algorithm to use is likely impacted by the problem state that agent faces. In coordination problems, these problem states represent the set of possible coordination problems between members of the group. To formalize this problem, let $S = \{s_1, \ldots, s_M\}$ be a group of M states representing all possible coordination problems agents within G may encounter. For every state, the values of $\mathcal{G}(CA_{a_j})$ - $\mathcal{C}(CA_{a_j})$ realized through using algorithms $\{CA_1 \ldots CA_k\}$ can potentially be different. Thus, the question is not simply which algorithm to choose for one specific problem instance, but which algorithm to choose in all potential problem states.

One possible approach is to loop through all possible actions an agent can choose, and for each state calculate the potential utility, $\mathcal{UT}^a(CA_{a_j})$, or the values of $\mathcal{G}(CA_{a_j})$ and $\mathcal{C}(CA_{a_j})$ from using each of the possible k algorithms. This information could then be applied within decision frameworks such as an MDP. In fact, work by Lagoudakis and Littman [38] demonstrates that the algorithm selection problem can be modeled as an MDP for certain types of problems. However, the assumption of this approach is that is possible to effectively quantify all possible state interaction within the algorithm set. This is not possible for most coordination problems as the state space is very large, and not necessarily discreet.

Other possible approaches involve performing no learning in ad-

vance and instead attempt to identify the best algorithm exclusively during run-time. For example, Allen and Minton [2] suggested running all algorithms $\{CA_1 \ldots CA_k\}$ in the portfolio for a short period of time on the specific problem instance. Secondary performance characteristics are then compiled from this preliminary trial to select the best algorithm. Gomes and Selman [23] suggested running several algorithms (or randomized instances of the same algorithm) in parallel creating an algorithm portfolio. However, assuming running each algorithm incurs costs $\mathcal{C}(CA_1) \ldots \mathcal{C}(CA_k)$, these approaches are likely to be ineffecient as the cumulative costs of running all algorithms are likely to be higher than the potential gain of finding even the optimal choice.

By using novel teamwork measures, we are able to more effectively address the coordination algorithm selection problem. Instead of studying a large set of possible agent **actions**, we use teamwork measures to evaluate the effectiveness of how selecting that algorithm will effect that agent's **interaction** within the group. Assuming differences in these measures can be found between the various coordination algorithms agents can choose, the complexity within the intractable MMDP model can be quickly reduced. Furthermore, learning which algorithm to choose can be done after a short learning period, allowing for effective online decisions. Several key advantages emerge from the simplified decision model used in the coordination selection approach in this thesis:

- A reduced space of actions. By defining all possible actions as the selection of algorithms $\{CA_1 \ldots CA_k\}$, we reduce the set of all possible actions to only the k coordination algorithms.

- A simplified performance measure. By creating novel teamwork

measures, we are able to equate the relative effectiveness of the algorithms $\{CA_1 \ldots CA_k\}$. This facilitates creating a relative ranking of each algorithm which can be used instead of a difficult to compute reward value.

- Clustered state spaces. We found that each algorithm is typically best for large groups of similar problems. As such, one does not have to study reward functions for each and every state S, but can create large clusters of similar problems. After these similar problems are identified, the important question becomes when does one transition between these clusters. All of the individual states found within each of the clusters no longer need to be stored.

1.1.3 Dynamic Coordination

Finally, this thesis presents dynamic coordination approaches that significantly improve performance over the static methods they are based on. To date, very few works exist that explored this idea. The concept of switching between groups of coordination methods was previously described as part of the TAEMS theoretical framework [39]. Their approach encodes tasks and potential relationships between agents. If any dynamics exist within the system, they must be formally encoded in advance. As such, their work concedes the necessity of preplanning or replanning for contingencies, making the system unable to adapt to runtime dynamics unless all possibilities have been formally modeled in advance. The work by Toledo and Jennings [17] present a framework where coordination mechanisms are selected at run-time based on the prevailing domain conditions. This is accomplished through abstracting the coordination problem as utility based functions that

can be reasoned about during task execution. This work is similar to ours in that we also create dynamic coordination methods that can operate online.

Several key points separate these approaches with the one presented here. First, the previous approaches [17, 39] require that interactions must be formally modeled, either in advance or during task execution. It is not clear how the formalized theoretical models can be transferred from the theoretical TAEMS statements or grid world domains they studied to real-world domains or actual groups of coordination algorithms. Similarly, it is unclear how one can take real-world problems and create tractable utility functions that quantify the utilities of the various coordination methods being used. We address these issues through the use of novel teamwork measures and simplified algorithm selection model. Finally, the approach of Toledo and Jennings [17] did not always improve the group's performance. In contrast, the dynamic approach we present was successful in consistently improving performance, typically by significant amounts beyond the static methods they are based on in a variety of domains.

1.2 Thesis Structure and Overview

The first chapter of this thesis addresses how one can create adaptive coordination methods for robots without communication. Many collaborative multi-robot application domains have limited areas of operation that cause spatial conflicts between robotic teammates. These spatial conflicts can cause the team's productivity to drop with the addition of robots. This phenomenon is impacted by the coordination methods used by the team-members, as different coordination methods yield radically different productivity results. However, selecting

the best coordination method to be used by teammates is a formidable task. This chapter presents techniques for creating adaptive coordination methods, to address this challenge: 1) We first present a combined coordination cost measure, CCC, for quantifying the cost of group interactions. This measure is useful for facilitating comparison between coordination methods, even when multiple cost factors are considered. We consistently find that as CCC values grow, group productivity falls. 2) Based on the CCC, we create adaptive coordination techniques that are able to dynamically adjust the efforts spent on coordination to match the number of perceived coordination conflicts in a group. We present two adaptation heuristics that are completely distributed and require no communication between robots. Using these heuristics, robots independently estimate their CCC, and adjust their coordination methods to minimize it, thus increasing group productivity. 3) We used a simulated robots to perform thousands of experiment trials, to demonstrate the efficacy of this approach. We show that using adaptive coordination methods creates a statistically significant improvement in productivity over static methods, regardless of the group size.

The second chapter studies how the CCC measure can be extended to address adaptation between robotic groups using communication. Communication between robots is one mechanism that can at times be helpful in such systems, but can also create a time and energy overhead that reduces performance. In dealing with this issue, various communication schemes have been proposed ranging from centralized and localized algorithms, to non-communicative methods. We argue that using the CCC measure can be useful for selecting the appropriate level of communication within such groups. We show that this measure can be used to create adaptive communication methods that switch

between various communication schemes. In extensive experiments in a robotic foraging domain, teams that used these adaptive methods were able to significantly increase their productivity compared to teams that used only one type of communication scheme.

In the third chapter we investigate methods for creating dynamic coordination methods in classic distributed constraint optimization problems. While these are NP-complete problems, many heuristics have been proposed for solving these problems. We found that the best method to use can change radically based on the specifics of a given problem instance. Thus, dynamic methods are needed that can chose the best approach for a given problem. Towards this goal, we present a dynamic algorithm selection approach where agents quantify the expected utility transmitting information will have. We found that large differences in this expected utility typically exist between algorithms, allowing for a clear policy with very short training periods. We present the results of thousands of trials within general graph coloring and TAEMS scheduling domains that demonstrate the strong statistical improvement of this dynamic method over the static methods they based on.

In the fourth chapter we study various coordination algorithms within distributed peer to peer search algorithms. Recent advances have presented viable structured and unstructured approaches for full-text search. We posit that these existing approaches are each best suited for different types of queries and present a hybrid system that leverages their relative strengths. Similar to structured approaches, peers within the network first publish terms stored within their documents. However, frequent terms are quickly identified and not exhaustively stored, resulting in a significant reduction in the system's storage requirements. During query lookup, peers rely on using unstructured

searches to compensate for the lack of fully published terms. Additionally, they explicitly weigh between the costs involved with structured and unstructured approaches, allowing for a significant reduction in query costs. We evaluated the effectiveness of this approach using both real-world and theoretical queries. We found that in most situations this approach yields near perfect recall. We discuss the limitations of this system, as well as possible compensatory strategies.

1.3 Publications

Subsets of the results that appear in this dissertation were published in the proceedings of the following refereed journals, book chapters, conferences and workshops:

Journal Articles

- Avi Rosenfeld, Gal A Kaminka, Sarit Kraus and Onn Shehory. A study of mechanisms for improving robotic group performance, accepted to *Artificial Intelligence Journal* (under revision), 2006.

Rigourous Refereed Conference Papers

- A. Rosenfeld, G. Kaminka, and S. Kraus. Adaptive robot coordination using interference metrics. In The Sixteenth European Conference on Artificial Intelligence, pages 910–916, August 2004.

Book Chapters

- Avi Rosenfeld, Gal Kaminka, and Sarit Kraus. A study of scalability properties in robotic teams. In Coordination of Large-Scale Multiagent Systems, pages 27–51, 2005.

- Avi Rosenfeld, Gal Kaminka, and Sarit Kraus. Adaptive robotic communication using coordination costs. In The 8th International Symposium on Distributed Autonomous Robotic Systems (DARS), pages 165–175, 2006.

Workshop and Other Proceedings

- A. Rosenfeld, G. Kaminka, and S. Kraus. A study of marginal performance properties in robotic groups. In Proceedings of the AAMAS 2004 Workshop on Coalitions and Teams, New York, 2004.

- A. Rosenfeld, G. Kaminka, and S. Kraus. Adaptive robot coordination using interference metrics. In Proceedings of the AAMAS 2004 Workshop on Learning and Evolution, New York, 2004.

- A. Rosenfeld, G. A. Kaminka, S. Kraus. A study of marginal performance properties in robotic teams. In Proceedings of AAMAS-04 (Poster), pages 1534–1535, July 2004. Short version of Coalition and Teams Workshop Paper.

- A. Rosenfeld, G. A. Kaminka, S. Kraus. Measuring the cost of robotic communication. In Proceedings of IJCAI-05 (Poster), pages 1734–1735, August 2005. Short version of DARS Paper.

- A. Rosenfeld, G. Kaminka, and S. Kraus. Adaptive robotic communication using coordination costs for improved trajectory planning. In AAAI Spring Symposium on Distributed Plan and Schedule Management, Stanford, 2006.

Chapter 2

A Study of Mechanisms for Improving Robotic Group Performance

2.1 Introduction

Groups of robots are used to enhance performance in many tasks [14, 20, 29, 65]. However, the physical environment where such groups operate often pose a challenge for the robots to properly coordinate their activities. Domains such as robotic search and rescue, vacuuming, and waste cleanup are all characterized by limited operating spaces where the robots are likely to collide [6, 20, 29, 65]. Thus while adding robots can potentially improve group performance, collisions are likely to become more frequent. To address this issues, a variety of collision avoidance and resolution techniques have been previously presented [6, 18, 20, 56, 66, 71]. However, no one method is best in all domain and group size settings.

Matching the best coordination method for a given robotic team and its operating domain is a formidable task. To date, several coordination frameworks have been suggested for reasoning about teamwork

and coordination [25, 39, 69]. One possible approach is to use decision theoretic models such as Markov Decision Processes (MDP) [60] within any of these formalized frameworks. This could potentially allow robots to choose the optimal coordination method as needed during task completion.

However, while each of these approaches has been shown to be effective under certain conditions, in many real-world applications the problem of making the optimal coordination decision is computationally intractable [60]. The inherent complexity in using these approaches demonstrates the necessity of creating novel approaches to effectively deal with real-world issues in a tractable fashion.

Our approach is to investigate a combined coordination cost measure, CCC, that quantifies the production resources spent due to coordination conflicts. We present this multi-attribute cost measure to quantify resources such as time and fuel each group member spends in coordination behaviors during task execution. The combined coordination cost measure facilitates comparison between different group methods. We found a high negative correlation between this measure and group productivity, allowing us to understand why certain groups were more effective than others.

This negative correlation between performance and CCC facilitates development of adaptive coordination methods. The key idea is that if robots dynamically reduce their CCC, group productivity will be improved. To demonstrate this, we create robotic groups which dynamically adapt their coordination techniques based on each robot's CCC estimate. Robotic agents calculate CCC estimates autonomously, by noting the frequency of events in which collisions are possible (and may or may not take place). This is done in a distributed fashion, and without any feedback from group members—no communication

is necessary.

We present two adaptive coordination methods suitable for homogeneous robots based on the CCC estimates. The first method of adaptation works by tweaking the parameters of a given coordination method to adapt it to the frequency of possible collisions. The second approach proceeds to dynamically self-select between a range of mutually exclusive coordination methods. In order to quickly adapt to a changing environment, we use weight-based heuristics by which every robot in the group is capable of quickly modifying its coordination method to match its estimated CCC.

We used a well-tested multi-robot simulator, Teambots [4, 5] to simulate groups of up to 30 robots engaged in both search and foraging tasks. We performed thousands of experiment trials, to demonstrate the efficacy of this approach, with various team sizes and compositions.

We found that these adaptive coordination approaches resulted in a statistically significant increase in group productivity in the domains we studied, even when faced with dynamically changing conditions. During task execution, different robots in the group were engaged in different coordination resolution behaviors. In fact, we found that the best form of coordination changes over the course of time, or as the task is being completed. Thus, various forms of coordination are likely to be needed at different times during task execution.

While we cannot guarantee the optimality of these heuristic approaches, the experiments demonstrate that this approach is effective in achieving a statistically significant improvement in productivity without a prolonged training period. We believe that this is likely to be needed in many robotic domains as environment dynamics and noise make traditional learning approaches difficult to implement.

2.2 Productivity Increases in Robotic Groups

This paper focuses on understanding the interplay between group coordination and productivity in robot groups. A closely related topic, of the scalability of labor, has been extensively studied within economics. According to the *Law of Marginal—or Diminishing— Returns*, as additional production resources are added, the additional productivity yielded as a result decreases [8]. The highest returns on production resources are from the first beginning of the production cycle. They then diminish with additional production expenditure, until at some point, it typically becomes economically impractical to add more production resources; the cost of additional production resources outweighs the productivity they add.

To date, there have been limited—and often conflicting—studies into how robotic team productivity scales with the addition of robots. Rybski et al. [65] demonstrated that groups of identical robots can exhibit marginal returns, with productivity curves resembling logarithmic functions. The first several robots in the groups they studied added the most productivity per robot, and each robot added successively less. However, they did not study group sizes larger than five robots. In contrast, work by Fontan and Matarić [66] found robotic groups operating within a robotic foraging domain contained a certain group size, a point they call "critical mass", after which the net productivity of the group dropped. Similarly, Vaughan et al. [71] also reported that adding robots decreases performance after a certain group size. The motivation for this work lies in understanding when coordination methods would be successful in consistently realizing marginal gains, and when one could expect to encounter a "critical mass" in their group size.

2.2.1 Group Differences in Performance

Our study begins with a simulated foraging domain, in which we investigate how robot productivity is affected as group size is scaled up. Foraging is formally defined as locating target items from a search region S, and delivering them to a goal region G [21]. The foraging domain is characterized by a limited area of operation where spatial conflicts between group members are likely to arise [18, 20, 21, 56, 65, 66, 71]. Many robotic tasks such as waste cleanup, search and rescue, planetary exploration, and area coverage share this trait.

We used a well tested robotic simulator, Teambots [4, 5], to collect data. We preferred using a simulator over performing experiments with real robots as it allowed us the ability to perform thousands of trials of various team sizes and compositions. The sheer volume of this data allowed us to make statistical conclusions that would be hard to duplicate with manually setup trials of physical robots. However, code created in the Teambots simulator has been shown to directly port to Nomad N150 robots; all behaviors and features found within the simulator can be equally applicable to these physical robots [4].

Using Teambots [5], we simulated a foraging environment measuring approximately 10 by 10 meters. There were a total of 40 target pucks within the field, 20 of which where stationary within the search area, and 20 moved randomly. Each trial measured how many pucks were delivered by groups of 1–30 robots from each of the coordination methods we studied within 9 simulated minutes of activity. To overcome any dependencies on initial positions, we averaged the results of 100 trials with the robots being placed at random initial positions for each run. Thus, this experiment simulated a total of 21,000 trials (7 groups $\times$ 30 group sizes $\times$ 100 trials per size) of 9 minute intervals.

The simulated robots we studied were identical but for their implementation of their teamwork coordination behaviors. Each robot had three common behaviors: wander, acquire, and deliver. In the *wander* phase, the robots originated from a random initial position, and proceeded in a random walk until they detected a resource targeted for collection. This triggered the *acquire* behavior. While performing this second behavior, the robots prepared to collect the puck by slowing down, and opening up their grippers to take the item. Assuming they successfully took hold of the object, the deliver behavior was triggered. At times the puck moved, or was moved by another robot, before the robot was able to take it. Once this target resource moved out of sensor range, the robot reverted once again to the wander behavior. The *deliver* behavior consisted of taking the target resource to the goal location which was in the center of the field.

We implemented a total of 7 coordination methods based on previously developed collision resolution and avoidance algorithms, and variations thereof. All algorithms operate without prior knowledge of the domain, nor communication. We chose to contrast coordination methods from this category to focus exclusively on issues relating to coordination resolution behaviors.

The implementation of the *Noise* method was included in the Teambots [5] package. Balch and Arkin [6] describe this method as a system of using repulsion schema any time a robot projects it is in danger of colliding. Robots then also add a noise element into its direction vector to prevent becoming stuck at a local minima.

Vaughan et al. [71] describe an algorithm that uses *Aggression* to resolve possible collisions by pushing its teammate(s) out of the way. They posit that possible collisions can best be resolved by having the robots compete and having only one robot gain access to the resource

in question. In our implementation of this method, for every cycle a robot found themselves within 2 radii of a teammate, it selected either an aggressive or timid behavior, with probability of 0.5. If the robot selected to become timid, it backed away for 100 cycles (10 simulated seconds). Otherwise it proceeded forward, executing the aggressive behavior. As robots choose to continue being "aggressive" or to become "meek" every cycle, the probability that two robots will collide in this implementation is near zero.

Similar to the Aggression group, the *Repel* group backtracked for 500 cycles (50 seconds) but mutually repelled using a direction of 180 degrees away from the closest robot. The *TimeRand* group contained no repulsion vector to prevent collisions. However, when robots sensed they did not significantly move for 100 cycles (10 seconds), they proceeded to move with a random walk for 150 cycles (15 seconds) once these robots. The *TimeRepel* also only reacted after the fact to collisions. Once these robots did not move for 150 cycles (15 seconds), they then moved backwards for 50 cycles (5 seconds).

Finally, we created two groups that lack any coordination mechanism. The *Gothru* group was allowed to ignore all obstacles, and as such spent no time engaged in coordination behaviors. This "robot" could only exist in simulation as it simply passes through obstacles and other robots. This group represents a theoretical group performance without any productivity lost to collisions. At the other extreme, the *Stuck* group also contained no coordination behaviors but simulated a real robot. As such, this group was likely to become stuck and lose all productivity when another robot blocked its path.

Figure 2.1 graphically represents the foraging results from these coordination methods. The X-axis depicts the various group sizes ranging from 1 to 30 robots. The Y-axis depicts the corresponding

average number of pucks the group collected averaged over 100 trials.

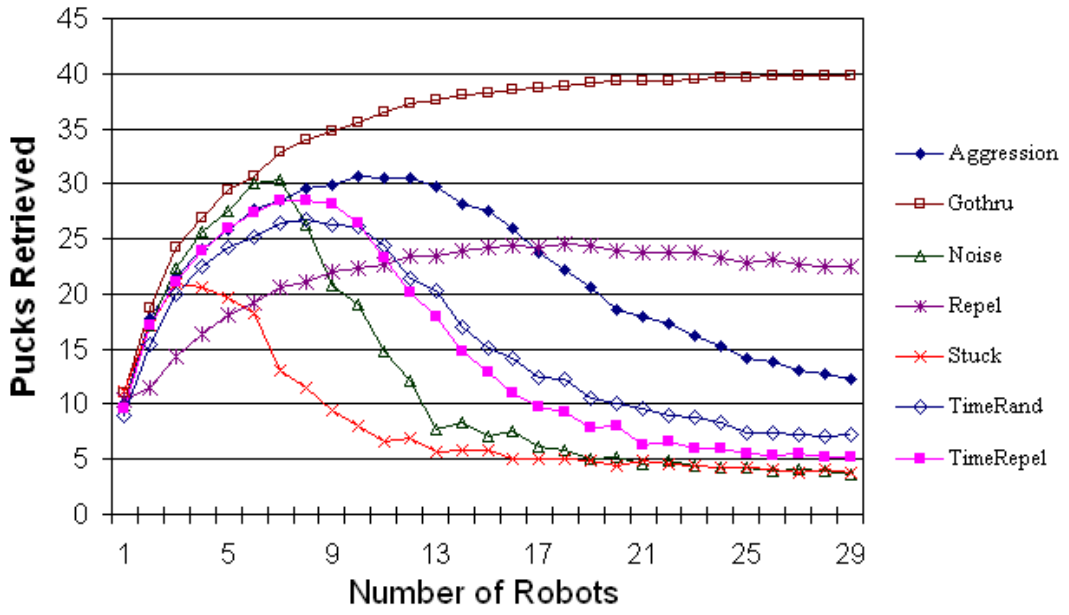

Figure 2.1: Motivating results comparing seven foraging groups. Each data-point represents the average pucks returned to the domain's home-base using that coordination method (Y-Axis) given that group size (X-axis).

According to economic theory, diminishing marginal returns are achieved when one or more production resources are held in fixed supply, while the quantity of homogeneous labor increases. In the foraging domain, the fixed number of pucks and limiting domain area acted as limiting factors of production. Consequently, one would expect to find production graphs consistent with economic marginal returns. However, only the theoretical Gothru group consistently demonstrated this quality over the full range of group sizes. All other groups contained a critical point where maximal productivity was reached. After the group size exceeded this point, productivity often dropped precipitously. For example, the Aggression group reached a maximum of 30.84 pucks collected in groups of 13 robots. Additionally, the coordination behaviors had a profound impact on each productivity level.

For example, when examining foraging groups of 10 robots, the Aggression method averaged over 30 pucks collected, the Noise group averaged approximately 20 pucks, and the Stuck group on average collected fewer than 8 pucks.

This research was motivated by these results. Based on this example, we focused on two open questions presented by Fontan and Matarić [18] and Arkin and Balch [3]:

- How one can determine when their coordination method has lost effectiveness? This may be useful for having robots shut down and save fuel, or to help maximize their productivity [18].
- What lessons can be learned about creating coordination methods to be resilient to dynamics in their environment. How can coordination methods be created that can quickly adapt to its domain conditions without a lengthy learning process [3]?

2.2.2 The Impact of Coordination on Robot Density

We propose that differences between coordination methods in spatially constrained domains can be explained based on robot density. As one adds robots into a domain, the density of robots, on average, should rise. Within spatially constrained domains, this can lead to certain area(s) having a bottleneck condition where robots cannot effectively complete their task, resulting in loss of productivity. However, having too low a density results in agents not reaching goal areas within the domain and thus not properly completing their task. As different coordination methods impact the group's density, it is critical that we properly match the coordination method to the domain conditions to achieve the best productivity for the group.

We can model robot density as follows: Let us pick a point p within a spatially constrained domain where a group of N robots must pass to complete their task. Given a radius r around this point, we focus on an area $A(r)$ surrounding p. During task completion, robots constantly move in and out of $A(r)$ with a certain heading α. At any given time t, there are k robots within any given area $A(r)$, where $k \leq N$. We denote the density, $\phi(r)$ as the total area of these k robots divided by the total area $A(r)$. The value of $\phi(r)$ will impact the group's performance. For example, $\phi(r) = 1$ indicates $A(r)$ contains no free space, and all robots mutually block. In these instances all productivity of the group will be lost until the area is cleared, and the density lowered. Conversely, assuming $\phi(r) = 0$, no robots are within the area. Assuming this value remains zero, no robots will complete their task, and the group's productivity will be zero until robots are allowed into the constrained area and $\phi(r)$ rises.

Figure 2.2 illustrates an example taken from the Teambots simulator with $k = 3$ robots within a radius $r = 1.5$ (meters). Note that we studied groups of homogeneous robots where each robot has a radius of approximately 0.25 meters. We denote the area of each robot as A', where $A' = 0.25^2\pi$ or 0.20. Thus, the density $\phi(1.5)$ as illustrated here would be $(kA')/A(r)$ or $(3 \times 0.20)/7.1$ or 0.08.

Every coordination method impacts the way in which robots prevent and resolve collisions, thus impacting $\phi(r)$. In general, coordination mechanisms that involve collision prevention behaviors well before robots collide will result in lower densities than methods that only trigger these behaviors once robots are closer. Similarly, methods that more aggressively space robots after collisions will result in lower densities than less aggressive methods. For example, a group whose coordination method requires robots to move away for a distance of 5

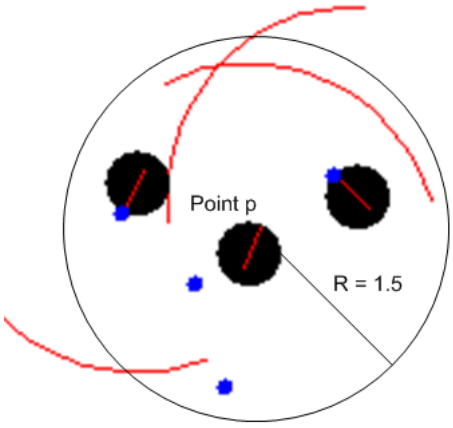

Figure 2.2: Three robots within $A(r)$ where $r = 1.5$. Picture is taken from the Teambots simulator and is drawn to scale.

meters after a collision will have a lower density than a method that only requires robots to move away 1 meter.

We claim that as robots are added or taken away from a domain, the best coordination method will change. When the group size (N) is small, the number of robots (k) coming within the constrained area is also likely to be small. In these cases, coordination methods should allow robots to complete their task uninhibited, and not further reduce $\phi(r)$. As N grows, k will naturally grow as well, and naive methods will result in too high values for $\phi(r)$. In these cases, methods that more robustly disperse the robots will be needed.

Determining the exact optimal value for $\phi(r)$ for a given domain and set of robots is an complex challenge, as many factors must be accounted for. First, we must model the speed of robots with regard to various domain conditions and behaviors. For example, the robots we studied slowed down to pick up objects, deviating from their maximal speed. Such phenomena must be exactly accounted for. Second, we must model the robots' exact positions and headings throughout task completion. In general, every robot heading towards p will have a velocity vector V_i based on its heading α from its initial position P_i

towards its final destination point p. For an exact model, every coordination method's response to different positions and headings must be precisely calculated. Finally, a simplified model assumes robots mutually block only in head on collisions. In fact, even indirect collisions also block robots, and thus the "collision area" of a robot needs to be modeled as $P_{i_{x+\epsilon}}$ and $P_{i_{y+\epsilon}}$ instead of the location P_i the robot is currently situated in. Given the complexity of modeling these different factors, we leave calculation of an optimal $\phi(r)$ for future work.

Nevertheless, we can generally demonstrate two important characteristics based on our model: (i) Differences in density exist between coordination methods; (ii) Given a certain radius r, some density value $\phi(r)$ results in the best group performance, regardless of the group size (N) operating within the domain, or the specifics of the coordination method used. The latter is a very important observation, as it may provide guidelines for matching coordination methods to specific domains based on their derived density. To demonstrate these claims, we logged the value of ϕ as a function of various distances r from the home-base (point p) within the foraging domain, and various group sizes. Specifically, we studied how values of ϕ corresponded between coordination methods taken at distances of $r = 0.5$, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, and 5.0. As was the case in Figure 1, we averaged every value from 100 simulated runs.

First, we compared the Aggression, Noise, Repel and Stuck coordination methods defined in the previous section. Recall from Figure 2.1 that the Aggression method performed best in groups of 10 robots, and Repel performed best in groups of 30. In Figure 2.3 we plot the density functions for $N = 10$ (the graph on the left) and 30 robots (the graph on the right). Note, that differences in coordination methods' densities were most pronounced when studying smaller distances for

r around point p. As one would expect, as $A(r)$ encompasses progressively larger portions of the entire domain area, the number of robots within this area (k) eventually equals N and no differences should be expected between coordination methods. Consequently, we only focus on density differences within small values for r. The Aggression method, which performed well in medium sized groups, did not successfully resolve conflicts in larger groups. This is reflected by an increase in density when moving from 10 robots to 30 robots. Conversely, Repel, which was effective in larger groups, exhibits a (too) low density in small and medium sized groups, reflecting a relatively lower productivity.

Second, when carefully inspecting the density levels for which the coordination methods have arrived at maximal productivity, it appears that some optimal density level exists. Specifically, one can observe that the density graphs for Aggression in groups of 10 and for Repel in groups of 30, are nearly identical (recall that these graphs correspond to methods performing optimally for a given group size). Inspecting the density values arrived at by these methods shows they are almost identical $\phi(0.5) = 0.18$, $\phi(1.0) = 0.15$, etc., from which we can conclude that optimal performance corresponds to a common density pattern. As we show below, other observations support this conclusion.

Similarly, one may question if the parameters within the coordination methods provide optimal densities. The Repel method we defined in the previous section backtracks for 50 seconds after a detected collision. We posit that different backtracking amounts would create different densities, each most appropriate for different domain conditions. To support this claim, we created variations of the Repel behavior where repel values of 5, 10, 20, and 50 seconds (Repel50,

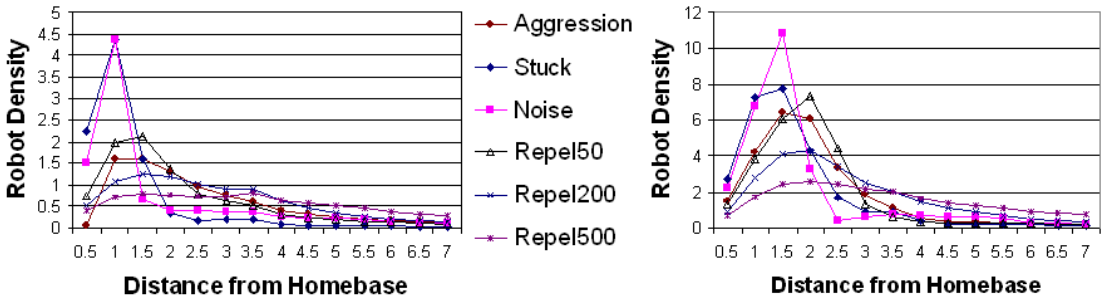

Figure 2.3: Robotic density for four coordination methods for groups of 10 robots (on left) and 30 robots (on right)

Repel10, Repel20, and Repel50 respectively) were used. Figure 2.4 displays these density functions for group sizes N =10, 20 and 30. Note that the density graphs of Repel50 in groups of 10, Repel200 in groups of 20, and Repel500 in groups of 30 are quite similar, and again reflect values similar to those seen in Figure 2.3. In fact, as we will see within the experiments sections (see Figure 2.10) these Repel values yielded the highest productivity in these group sizes.

We believe that the model could theoretically be used to calculate an optimal density for a given domain. A group designer could then compare the coordination methods at her disposal, and select the one closest to this optimal density. Furthermore, this model may also give us insight into predicting when the productivity of a group will be, and the amount a specific coordination mechanism deviates from the theoretical optimal performance level. For example, if one would know the density needed to achieve optimal performance, one could adjust the repel values within this coordination method to ensure that this condition is met.

However, this paper's assumption is that the number of variables involved with creating this precise model, and their associated states, makes determination of the optimal density impractical, for this and most real-world settings. Instead, we focus on developing a CCC mea-

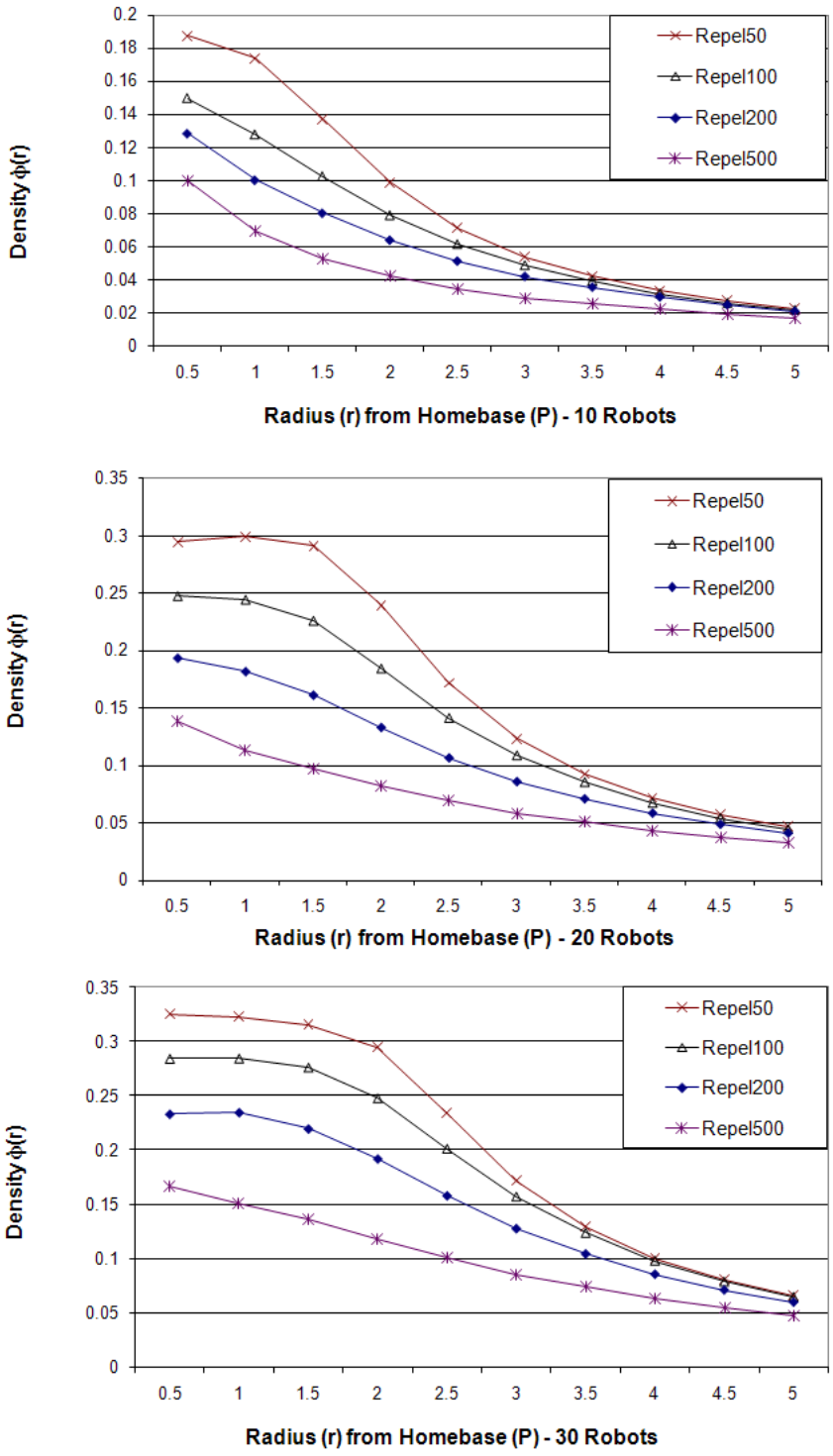

Figure 2.4: Comparing robotic density for coordination methods Repel50, Repel100, Repel200, and Repel500 in groups of 10 (top), 20 (middle) and 30 (bottom)

sure that is significantly easier to calculate and can be autonomously measured by each robot. This measure requires no prior knowledge

of the specifics of the coordination methods being used, or a-priori knowledge of domain parameters. Nonetheless, as the next section demonstrates, this measure is still effective in modeling differences in resources spent on resolving coordination conflicts. Furthermore, as sections 2.4 and 2.5 demonstrate, this measure can also be used to create adaptive methods that quickly and effectively adapt the coordination of the team to the task.

2.3 Quantifying the Cost of Coordination: the CCC Measure

A mechanism is needed to measure why certain coordination mechanisms are more effective than others. In this section we present such a measure of coordination, the Combined Coordination Cost measure (CCC). We find that this measure and productivity are strongly correlated, and use this measure to explain differences in productivity between all teams. As one might expect, the more efforts the group spends in coordination behaviors, its ability to complete the task at hand is diminished. We posit that in the absence of coordination conflicts such as those caused by spatial conflicts, all teams should consistently demonstrate marginal gains during scale up. We confirm this idea by easing the spatial conflicts inherent in the domains and note that all groups consistently demonstrate increasing marginal productivity returns.

2.3.1 Measuring Combined Coordination Costs

The CCC is defined as the sum of resources a group member expends because of its interactions with other members, in particular resolving conflicts between agents (preventing conflicts and managing their con-

sequences). Examples of these resources may include the time, fuel, and money spent in coordination activities or in any combination of factors. Each agent expends a coordination cost $\mathcal{C}_i$, that impacts the entire group's productivity. This cost can consist of multiple factors, $\mathcal{C}_i^j$, with each one containing a relative weight of $\mathbf{P}_j$. We create a multi-attribute cost function based on the Simple Additive Weighting (SAW) method [77] often used for multi-attribute utility functions.

We describe the combined coordination cost of a specific agent as follows. Let $G = \{a_1, \ldots, a_N\}$ be a group of N agents engaged in some cooperative behavior. Let $\mathcal{C}_i = \{\mathcal{C}_i^j\}, 1 \leq j \leq t$ be the set of t coordination costs in the system derived from the actions of agent a_i. Let $\mathbf{P}_j$ be the ratio of each factor of $\mathcal{C}_i$ in the total cost calculation, i.e., $\sum_{j=1}^{t} \mathbf{P}_j = 1$. As the total coordination cost of each agent is the simple weighed sum [77] of all of these costs, the final cost equation is:

$$\mathcal{C}_i = \sum_{j=1}^{t} \mathcal{C}_i^j \cdot \mathbf{P}_j \tag{2.1}$$

In contrast to Goldberg and Matarić *interference* measure, [20] we model resources spent in coordination even before a specific conflict, such as robotic collisions, occurs. For example, the Aggression group's timid and aggressive behaviors to avoid collisions all constitute coordination costs by our definition. The TimeRand and TimeRepel groups have costs only after a collision is detected. The Gothru group's CCC measure was always zero because it never engages in any collision avoidance resolution behaviors and thus represents idealized group performance.

According to the hypothesis, we expected to see a negative correlation between CCC measures and productivity, in two major respects.

First, the degree to which a group deviates from idealized marginal gains is proportional to the average CCC level within the group. This in turn impacts the group size where the group reaches its maximal performance. Second, even before groups hit their maximum productivity point, we hypothesized that the more productive groups have lower CCC levels than their peers. This accounts for the varying productivity levels in equally sized groups.

2.3.2 Measuring CCC from Various Resources

In order to confirm this hypothesis, we reran the seven foraging groups and logged their average CCC levels. Note that in the first study, as is the case in the work of Goldberg and Matarić [20], we only consider the $\mathcal{C}_i$ for time spent on coordination. With the exception of the Gothru which never registered any costs, we used the simulator to measure the time associated with the robots' collision avoidance and resolution coordination behaviors. For all groups other than the Stuck and Gothru groups, we additionally measured the time the robots used collision resolution behaviors when they were not colliding. In the Noise and Repel groups, this represented the time spent in repelling activities. In the Aggression group, it was the time spent in timid and aggressive behaviors. In the Timeout groups, this was the time spent trying to resolve a collision once the robot timed out. Figure 2.5 represents the result from this trial. The X-axis once again represents the group size over the 1–30 robot range, and the Y-axis represents the average time that each robot within the group spent in coordination behaviors (out of 540 seconds) over the 100 trials.

Overall, we found a strong negative correlation (average -0.94) between groups' performance and their CCC levels, in all groups sized

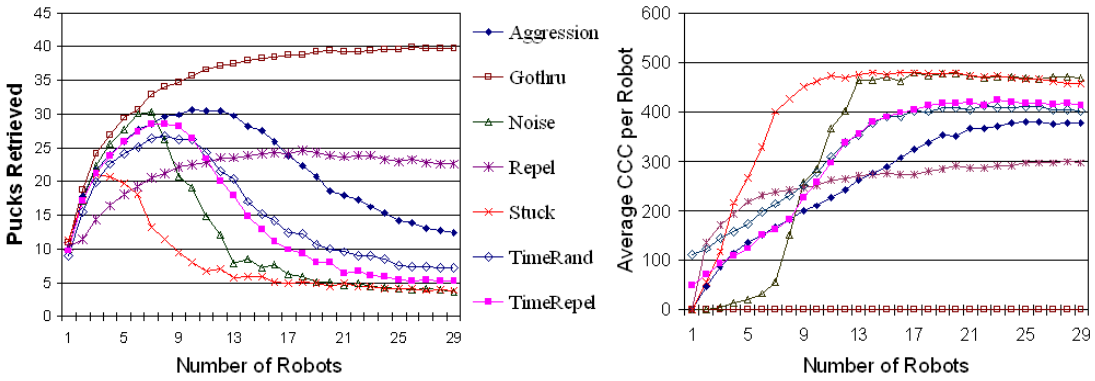

Figure 2.5: Comparing foraging groups' coordination costs

1 to 30 robots. The lower the average robots' coordination cost, the higher that groups' average productivity. The intuitive explanation is that since the task was bounded only by time, the more time spent on coordination behaviors, the less time was available for properly completing the task. Thus, groups that minimized this cost were more effective.

However, the CCC measure is also capable of taking other costs into consideration. We also implemented these same coordination methods, but used fuel instead of time as the one limiting production resource, i.e. $\mathbf{P}_1 = 1$ again. In this experiment we allocated each robot 300 units of fuel. We assumed the fuel used was proportional to the distance traveled, with a much lower amount of fuel (1 unit per 100 seconds) consumed for basic robot sensing and computation. Fuel was not transferable. Once a robot ran out of fuel, it stopped functioning and became an obstacle. Once again, we reasoned that certain methods would be more successful than others in minimizing this measurement under varying domain conditions.

Figure 2.6 graphically presents the foraging productivity results over the group range of 1–30 robots when only accounting for coordination cost based on fuel. We again found a strong negative cor-

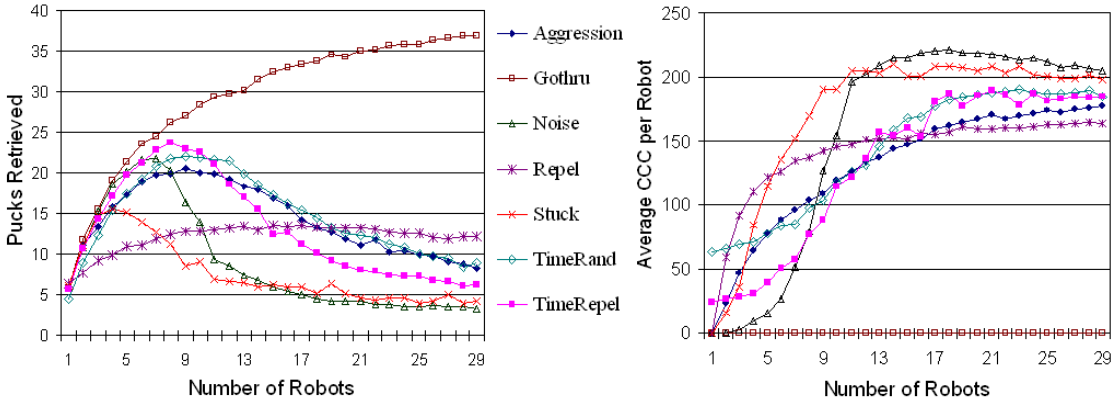

Figure 2.6: Comparing group productivity and coordination fuel cost measures in foraging groups

relation (average -0.95) between the coordination cost at the agent level, and the group's productivity. Notice that the cost functions of these method are effected by the new domain requirements (productivity bounded by fuel instead of time) and the ordering of the best coordination methods changes as a result. In these trials the Timeout based groups (TimeRand and TimeRepel) fared best in medium sized groups, while these groups never had the highest productivity in the first set of experiments.

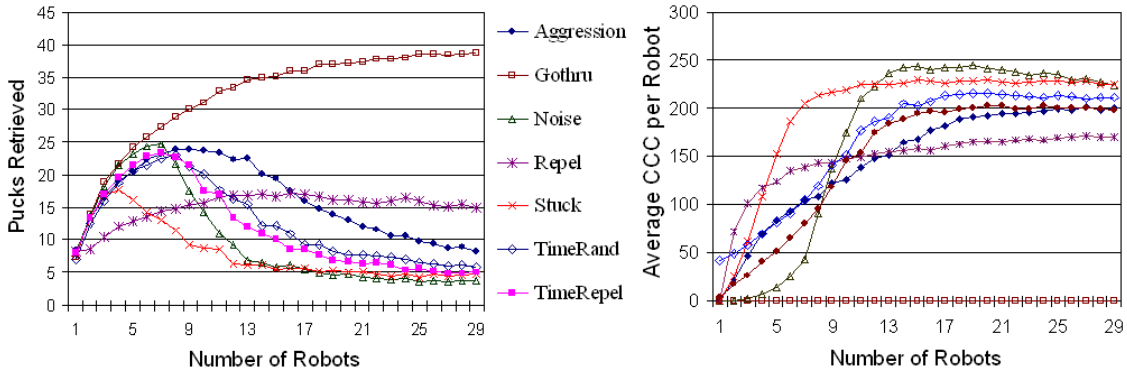

Figure 2.7: Comparing group productivity and multi-attribute coordination cost measures

Realistically, some combination of production resources are likely to bound an agent's productivity. As a result, we also studied cases of multi-attribute cost functions, and present the results for $\mathbf{P}_{Time}$

= 0.7 and $\mathbf{P}_{Fuel} = 0.3$. While time and fuel are different resources, we created a combined cost function by viewing the cost $\mathcal{C}_i^{Time}$ as a constant amount of fuel that was detracted every second of the robot's operation, independent of its movement. This allowed us to normalize the time cost to approximately 70 percent of the total cost function and create a cost function composed of these two factors. Figure 2.7 presents the results for this multi-cost attribute function, with the lower Y-axis here measuring the combined cost of both factors, out of 300 total units. The multi-attribute measurement was still strongly negatively correlated (-0.94 on average) to each group size and its corresponding average productivity level.

The CCC measure is equally applicable to other domains as well. To demonstrate this claim, we studied a spatially limited search domain constructed as follows. Using the Teambots [5] simulator, we created a room of approximately 3 by 3 meters with one exit 0.6 meters wide and placed groups of robots inside (for comparison purposes each robot is approximately 0.5 meters wide). We measured the time until the first robot found a target item outside the room. We ran trials of groups of six out of seven coordination methods (the Gothru method is not applicable to search tasks) in sizes from 1–23 robots (the room holds a maximum of 23 robots) and averaged the results from 50 trials. We measured the coordination cost in terms of the time and/or fuel used per robot in coordination behaviors while accomplishing this task.

We again found a high correlation between the cost measurement based on the robot's time spent in resolving conflicts, and the total time it took for the group to complete its task. We first considered the case of only the time cost being important ($\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$). We capped each experiment at 15 minutes of activity, after which we assumed the task could not be completed by that group. The results

from this experiments are presented in Figure 2.8. In the left portion of the graph, we display the time length (in seconds) until the task was completed as the Y-axis with the X-axis showing the different group sizes. We found that most groups were able to complete their task more quickly with small groups of robots. After some group size, we again found that adding additional robots detracted from the group's overall productivity. The right graph displays the average CCC measurement based on time alone. The Y-axis depicts the number of seconds (out of 900 seconds) the robots were engaged with, on average, dealing with spatial conflicts. As the robots spent more time resolving group conflicts, more time was needed to complete the task.

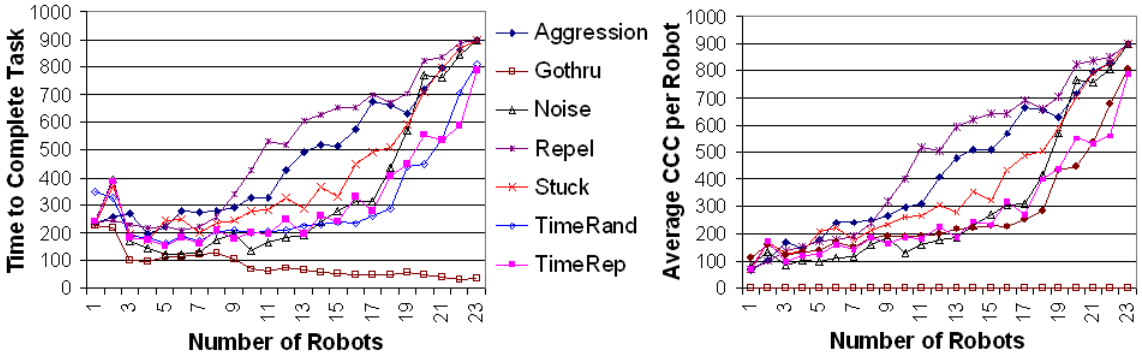

Figure 2.8: Comparing group productivity and coordination time cost measures in search groups

We found a very high correlation (average 0.97) between the average measurement of each robot's time cost measurement, and the time to complete the task. Note that in this domain, lower search times are better, thus higher productivity is represented by lower values. Therefore, the high correlation in the search domain is positive, while it was strongly negative in the foraging domain. Still, in both cases, as the CCC measure increased, the group's productivity decreased.

The relationship between coordination costs in energy based cost measures and multi-attribute costs also applied to this new domain.

In the experiments where $\mathbf{P}_{Time} = 0$ and $\mathbf{P}_{Fuel} = 1$ we allotted each search robot with 300 units of fuel. As was the case in the foraging domain, the robots used this fuel to move, but also used a smaller amount to maintain basic sensors and processing capabilities. We also created experiments for $\mathbf{P}_{Time} = 0.7$, and $\mathbf{P}_{Fuel} = 0.3$ with the same standardization between time and fuel as found in the foraging domain. The fuel-only experiments had a correlation of 0.99 between the fuel used in resolving conflicts, and the average fuel used until the first robot completed the task, while the equivalent weighted experiments had a correlation of 0.98. As opposed to the foraging domain, the ordering of the most effective coordination methods was not effected by the cost functions of $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$, or $\mathbf{P}_{Time} = 0$ and $\mathbf{P}_{Fuel} = 1$, or $\mathbf{P}_{Time} = 0.7$, and $\mathbf{P}_{Fuel} = 0.3$. In all cases, the Noise group had the best time to task completion and the lowest fuel usage to task completion in small groups. The TimeRand group had the best time to complete the task and the lowest fuel usage in larger groups. This result is intuitive, as many domains exist when fuel usage and time to task completion are correlated.

Thus, in both domains the CCC measure was successful in predicting the relative effectiveness of coordination methods. In the foraging domain the correlation between the group productivity and this measure ranged from -0.94 to -0.96. In the search domain it was even slightly higher, and ranged between 0.97 and 0.99.

2.3.3 Coordination Conflicts: The Trigger for Large CCC Values

According to the density model, different coordination methods affect robots' interactions within spatially constrained domains and the goal

must be to properly match the best coordination method to the needs of the domain. Care must be taken not to spend too much on coordination, and thus unnecessarily lower the group's density, or too little, and thus have too high a density. Robots with too low a density have spent too much preventing collisions. If robots have too high a density, they have not spent enough on coordination and will constantly retrigger collision resolution behaviors too quickly.

The CCC measures this expenditure of the resources spent before and after coordination conflicts. It is for this reason that the CCC can effectively measure (after the fact) which method in total spent the least on coordination, and thus achieved the best density and highest productivity.

However, the goal is also to develop mechanisms to improve group performance. In order to do so, the robots must be aware of the conflicts that trigger coordination resolution behaviors. In this section we demonstrate that the spatial conflicts inherent in the domains we studied triggered the CCC costs. Once we removed the reason for conflicts, groups consistently achieved marginal gains, and differences between coordination methods became less pronounced.

Within the foraging domain, spatial conflicts revolved around the one home-base within the operating area. We modified the foraging group requirement of returning the pucks to one centralized home base location. Instead, robots were allowed to deposit their pucks as soon as they picked them up, without returning them to any one location. We left all other environmental factors such as the number of trials, the size and shape of the field and the targets to be delivered identical. Teambots [5] was again used tot simulate 21,000 trials (7 groups $\times$ 30 group sizes $\times$ 100 trials per size) of 9 minute intervals in this experiment.

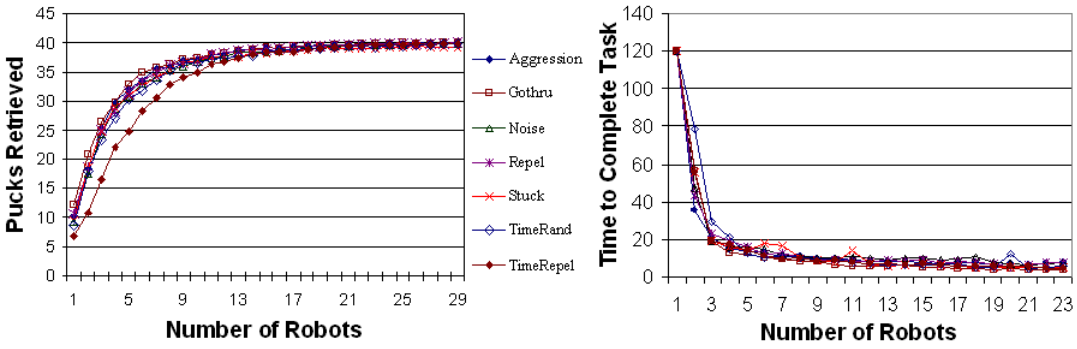

Figure 2.9: Modified foraging and search domains

As the left side of Figure 2.9 shows, all groups did indeed always achieve marginal returns in the modified foraging domain. While Gothru still performed the best, the differences between it and other groups' coordination methods were not as pronounced. Most groups had very similar coordination costs, and also productivity levels. The exception was the RepelRand group which had relatively high costs in small groups, and also lower performance. However, even this group consistently demonstrated marginal gains in productivity as the group size grew.

Within the search domain, we hypothesized that limitations in the room size and width of the exits created coordination costs during scale up. In order to ease this restriction, we doubled the size of the room to become approximately 3 by 3 meters, and widened the exit to allow free passage out of the room by more than one robot. Once again, we measured the time elapsed (in seconds) until the first robot left the room and averaged 100 trials for each point. This experiment also constituted nearly 14,000 trials (6 groups $\times$ 23 group sizes $\times$ 100 trials) of varying lengths. The right side of Figure 2.9 graphically shows that the modified domain consistently realized marginal increases in faster search times with respect to group size. Once again, cost levels were also negligible in the new domain. Thus, we concluded that achieving

marginal productivity gains was always possible once competition over spatial resources was removed.

2.4 Improving Productivity through Coordination Metrics

In this section, we demonstrate how the CCC measure is useful for helping robots self-evaluate the effectiveness of their coordination methods in an online fashion. By monitoring the triggers of coordination conflicts, robots are able to adapt their coordination methods to the needs of their environment. Robots that use such an approach demonstrate a statistically significant improvement in productivity over non-adaptive methods.

The dynamic nature of robotic environments makes the challenge of creating adaptive coordination formidable. While traditional reinforcement learning methods have been used within some robotic environments [45, 50], the number of iterations such algorithms require makes them unproductive without a significant training period. Even after robots could learn the theoretically optimal coordination method for their specific environment, events such as changes in the environment or hardware failures would likely render these policies obsolete. Furthermore, finding the optimal coordination method for a group is even a harder problem, with typically intractable complexity [60]. This is because the state-space of all possible possible actions, taken together with all possible interactions, is of exponential size. As such, even without any dynamics within the system, finding the optimal coordination is not always feasible.

We therefore focus on using CCC heuristically, to allow robots to dynamically select coordination algorithms during task execution. The

approach requires no prior knowledge of the domain's physical dimensions, boundaries, number of obstacles, or number of other teammates. The possible state-space is limited to mapping values of CCC to the coordination methods at the group designer's disposal—a tractable problem that can be quickly addressed.

We present two adaptive coordination methods and their advantages above static methods. In the first technique we have the robots self adjust parameters within one coordination method to match the perceived environmental conditions. The second technique involves adaptation between a number of distinct and mutually exclusive, coordination methods. We found that both approaches did indeed significantly outperform the static methods we studied in both the foraging and search domains.

2.4.1 Adaptive Coordination Algorithms

The adaptive approaches we present are based on having each robot maintain an estimate of local coordination conflicts. This estimate is adjusted as collisions occur and / or are resolved and is thus sensitive to the triggers of the CCC costs. Specifically, the algorithm works as follows: Every robot autonomously measures its own estimate, V to represent the likelihood coordination conflicts are about to be encountered. We first initialize V to a base value, V_{init}. For each cycle that passes where that robot detects no impending collisions, it decreases its value of V by a certain amount, W_{down}. For each cycle where a robots sense a collision is likely, it increases its value V by a certain amount, W_{up}. This process continues autonomously for all robots within a group. Furthermore, this process does not require any communication between group members. Thus, it conceivable, and

even likely, that robots will have different values for V based on the localized conditions it is currently encountering.

The value V is pivotal for determining the coordination method to be used. When V is low, the robot has resolved all coordination conflicts and should use methods with low coordination overhead that do not further lower the group's CCC. This allows the robot to finish its task as quickly as possible. When conflicts are more common and V is high, more costly methods are needed to reduce the group's density. This removes a potential bottleneck condition, allowing some of the robots to complete their task within the spatially constrained area.

In the first group of adaptation methods, we translate values for V directly as a parameter of the coordination method. For example, we use this value to determine the number of cycles the *Repel* method uses to repel once it detects a collision is imminent or the time period chosen by the *TimeRand* method before engaging in collision resolution behaviors. This way, each robot can autonomously control the strength of their Repel resolution behaviors.

In the second adaptation method the values for V are used to switch between a set of coordination techniques that have been pre-ordered based on their coordination overheads as ranging from simple to complex ones. Ranges of values for V are then mapped to these mutually exclusive methods. V_{init} corresponds to the starting point represented by the coordination method with the lowest overhead, and the values of V_{up} and V_{down} are then used to change the robot's fundamental coordination mechanism. Once the value V rises or falls below a certain threshold, that robot will change its fundamental coordination method as needed.

2.4.2 Quickly Setting the Weight Values

We now discuss how the weights, V_{init}, W_{up}, and W_{down} can be quickly set. It is important to stress that these weights form an approach to resolving coordination conflicts online. Our goal is not to find any one optimal coordination method as we found that dynamics within the domain require different coordination methods throughout the task completion. For example, assume one robot ceases functioning in the middle of the task, it may be required to switch coordination methods because of this event. Thus, the goal is to find a theoretical policy, π, based on the robot's estimate V that can be used to change the coordination method each agent uses in an optimal fashion.

While traditional learning methods, such as Q-learning [72] and other methods [73, 68] guarantee the ability to find an optimal policy, there are several major challenges in implementing this approach here. The first is procedural. Q-learning is based on a Markov based decision process that requires a concept of "state" that is difficult to define during task execution. As opposed to clearly defined discrete domains, there is no reward for any given cycle of activity in the robotic domains we studied. Thus, the ability to evaluate the effectiveness of any given action can only be done after a relatively long trial. This in turn leads to a second problem—namely the amount of exploration data typically needed in Q-learning and other traditional learning methods to converge on an optimal solution. The thousands or hundreds of thousands of trials that might be needed are impractical for physical robot trials [37]. For example, in the foraging domain previously mentioned, we studied 7 groups of coordination methods over group sizes of 1–30 robots. Each productivity data point was averaged from 100 trials for statistical significance, or a total of 21,000 trials. Third, even if

a theoretical optimal policy might be found dynamics within robotic domains may render these policies obsolete very quickly and a new learned policy π would need to be created. Finally, even if some form of learning could produce optimal weights for V_{init}, W_{up}, and W_{down}, there is no guarantee that these weights form the optimal coordination policy for the group. This is because the robots' sensors yield only a partial observable picture of their environment, and make no use of communication to attempt to complete that picture. Work by Pynadath and Tambe [60] demonstrated that finding an optimal policy in such cases is NEXP-complete.

As a result, the goal is *improved* productivity through a adaptive policy over the static methods upon which it is based, which may or may not form *the* actual optimal policy. Our approach is to facilitate autonomous adaptation based on the CCC measure. This measure can be locally estimated without communication, and can be used for quickly achieving significant productivity gains without a prolonged learning period.

Previous work by Kohl and Stone [37] contrasted Hill Climbing, Amoeba, Genetic Algorithms, and Gradient Learning algorithms to learn improved walking speeds in quadruped Sony Aibo robots. Their problem has certain similarities to the weight assignment problem. On one hand their challenge involved attempting to converge upon an optimal weight values for 12 parameters in the robots' gait—as opposed to only 3 weights in our problem. However, their evaluation problem was considerably more simple–evaluating the average speed of one robot. In contrast, we search for an optimal weight policy that can react to dynamics such as changing group sizes. Thus, we need to evaluate the policy over a range of different group sizes, significantly complicating the process.

Similar to work by Kohl and Stone [37], we used two different learning approaches for setting the weights: Hill Climbing and Gradient Learning. For each learning method, we used two different types of evaluation functions. In one possibility, the average productivity from the entire range of robot group sizes was considered. As the coordination adaptation methods are intended to work for any group size, when evaluating the effectiveness of π, the average productivity from the entire group range should be calculated. The downside of this approach is the number of trials required for policy evaluation. Assuming 5 or more trials are needed for each data point, due to the noise common within robotic productivity in any one given trial, even evaluating a range of 30 robots requires 150 trials–a number of trials that would be difficult to perform once, let alone multiple times to converge on an optimal value. As a result, we also used an evaluation function that analyzed a selective group sampling of each policy. According to this approach representative group sizes are used to evaluate the new policy. In the experiments, we analyzed representative groups of small, medium and large group sizes. We selected the end point (group sizes of 2 and 30) as well as the middle group size (15 robots). We believed this would provide a reasonable estimate over the entire range with much fewer trials needed to evaluate any given policy. Various variations of this idea are possible such as randomly selecting the representative group size for evaluation from within set group range, learning the best group sizes to evaluate, and various heuristics. We leave the development of these ideas for future work.

In both of the algorithms, we set the initial π to approximate the parameters of the static coordination that served as a basis for adaptation. Any static coordination method could be viewed as containing a π with fixed values of V_{init}, W_{up} and W_{down}. One naive way of im-

proving on any static method is to choose random values for W_{up} and W_{down} which should improve performance beyond this point. For example, assume one is trying to create an adaptive Repel method based on a static method that repels for 200 cycles after a projected collision. Once one sets V_{init} to 200, any naive values of W_{up} and W_{down} should represent an improvement from this point. In the second type of adaptation, V_{init} similarly could be set to represent the method with the highest average productivity. Again, any changes resulting from W_{up} and W_{down} should only help after this point. Our Hill Climbing and Gradient Learning algorithms were then used to refine the weight values from this baseline.

Hill Climbing algorithms have the advantage that they are intuitive for this and similar parameterization problems [37]. In this method, random perturbations for the values of V_{init}, W_{up}, and W_{down} are evaluated. If these values represent an improvement in the group's overall productivity, judged through either of the two methods evaluation functions previously described (either average sampling over the entire range, or selective sampling), these new values are accepted for π. Otherwise, the changes are discarded. The following pseudo-code describes the approach:

Algorithm 1 Hill Climbing

1: $\pi \Leftarrow$ Initial Policy (as described in paper)
2: **while** not done **do**
3: Create variation of π policy, π_{new}, with random perturbations in V_{init}, V_{up}, and V_{down}
4: **if** Productivity(π_{new}) > Productivity (π) **then**
5: $\pi \Leftarrow \pi_{new}$

Our Gradient Learning implementation is built upon the Hill Climbing approach. In both cases, perturbations in values for V_{init}, W_{up}, and

W_{down} are created and evaluated. However, in this approach, each change is evaluated individually. Instead of simply accepting a change as is, a function of the improvement caused by this factor is accepted. In the experiments, we used a normalized value in the change times a small constant, or

$$\Delta(|V_{New-weight} - V_{Old-weight}|)/V_{Old-weight} \times Constant \tag{2.2}$$

to create a normalized gradient direction. The following pseudo-code describes this algorithm:

Algorithm 2 Gradient Learning

1: $\pi \Leftarrow$ Initial Policy (same as in approach #1)
2: **while** not done **do**
3: generate small variations for each parameter in the value of +/- ε Specifically:
4: Generate an ϵ change (perturbation) in parameter V_{init}
5: Evaluate new V_{init} policy
6: Generate an ϵ change (perturbation) in parameter V_{down}
7: Evaluate new V_{down} policy
8: Generate an ϵ change (perturbation) in parameter V_{up}
9: Evaluate new V_{up} policy
10: Create a new π policy based on gradient learning based on the combined evaluation of all three sub-policies. Specifically:
11: $\pi \Leftarrow$ old π modified with normalized gradient changes in all three parameters

As the next section details, both learning approaches were effective in significantly improving productivity over non-adaptive methods.

2.5 Adaptation Experimental Results

In this section we present the results in applying both adaptive approaches within the foraging and search domains we studied. The first type of adaptation, parameter tweaking within one method, was

effective in raising productivity levels to the highest levels of the static levels they were based on. Adaptation between methods was even more successful and often significantly exceeded the productivity levels of the static methods they were based on, especially in the foraging domain.

Section 2.5.1 presents the results of both of these adaptive methods in the foraging domain, and Section 2.5.2 discusses the respective results in the search domain. We also found that there was some flexibility in setting the weights and near "out of the box" productivity improvements were found. As we demonstrate in Section 2.5.3, even suboptimal weight values were still successful in significantly improving a group's performance. Finally, in Section 2.5.4, we present support for why the approach is so successful. We attribute the success to the robots' ability to quickly and effectively change coordination approaches based on their localized conditions in the dynamic environments they operate.

2.5.1 Adaptation in Multi-Robot Foraging

The first type of adaptation uses each robot's estimate of its CCC to adjust the strength within one given coordination method. In order to demonstrate the efficacy of this approach, we began by analyzing the strength of coordination behaviors within the *Repel* and *TimeRand* coordination methods previously mentioned. In the previous experiments we chose a length of 500 cycles (50 seconds) with the Repel group to move away from a robot nearing a collision. Our TimeRand group waited 10 seconds before considering itself stopped by another robot. As we described in subsection 2.2.2, these parameter values are likely to be optimal only for certain group sizes. Once again, the

optimal density, and thus the amount each robot spends in these behaviors, must be properly matched to the group size and needs of the domain. For example, if a Repel robot repels for too long after a potential collision, it will take longer to complete its task. However, in situations where collisions are likely to occur, too short a repulsion period results in too high a density, and robots will become stuck within the spatially constrained domain. A similar problem exists in the TimeRand group. If the timeout threshold is set too low, the robots will consider themselves inactive even while performing necessary tasks such as slowing down to attempt to take a target puck. Too long a timeout threshold results in inappropriately high densities, and robots will become stuck for long periods before attempting to resolve conflicts.

To demonstrate this phenomenon, we studied 5 variations of the Repel groups, choosing values of 10, 50, 100, 200, and 500 cycles as the length of time robots repelled after projected collisions. We found that the best variation of the Repel coordination method depended on the size of the group. As the group size grew, robots collided more frequently, and increasingly more aggressive coordination methods were needed to lower the group's density. Among the Repel groups, Repel50 had the highest productivity in the groups up to 10 robots. Between 10 and 15 robots the Repel100 group did best. The Repel200 group fared better over the next 5 robots, and the Repel500 group had the highest productivity between 20–30 robots. Overall, the Repel200 fared the best with an average productivity of 23 pucks. However, this group only fared the best over a range of 5 robots. The left side of Figure 2.10 represents the productivity of these static methods.

We proceeded to create an adaptive Repel groups where each robot used its CCC estimates to autonomously choose which repel value to

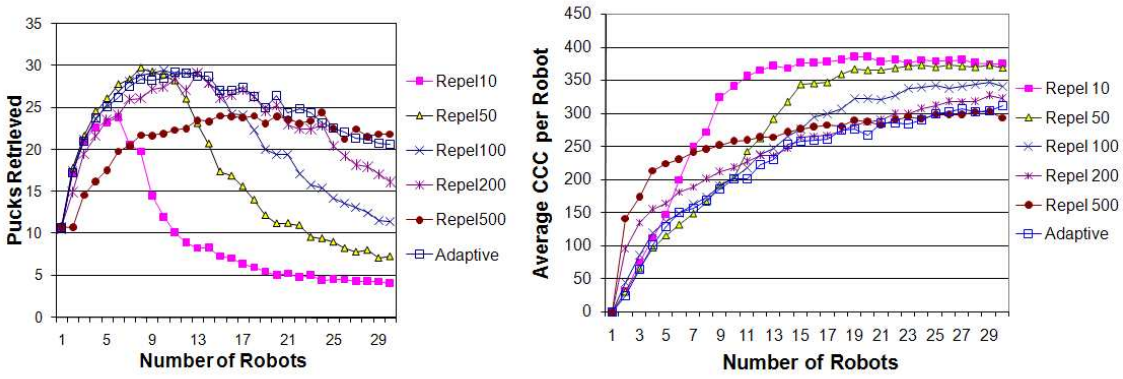

Figure 2.10: Productivity graphs in Repel (left) and CCC measure for all groups (right). Each data-point represents average productivity levels taken from 50 trials.

use. The left side of Figure 2.10 also displays the productivity results from the Hill Climbing Repel adaptive algorithm and coordination costs $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$. These results were taken after 5 learning iterations using the first evaluation function (taking the average productivity from 5 trials over the entire possible robot population). Similar results were obtained from learning trials of the other learning variations. Notice that the adaptive method often matches the highest productivity levels from the static groups. For statistical significance we ran all Repel groups for 50 trials over a range of 1–30 robots.

In order to evaluate the significance of these results, we conducted a two-tailed paired t-test on the data. We first compared the averaged productivity values of the adaptive Repel group to all of the non-adaptive methods over the range of 30 robots. All scores were far below the 0.05 significance level with the highest p-value for the Null hypothesis being only 0.00013 (between the adaptive group and the Repel 100 group), strongly supporting the hypothesis that this adaptive method statistically improved results over static methods.

The right side of Figure 2.10 demonstrates the success of this adap-

tive Repel group in minimizing coordination costs. The X-axis in this graph represents the group size, and the Y-axis corresponds to the robot's average coordination cost as measured in seconds. The adaptive group consistently registered the lowest coordination costs. In fact, the statistical correlation between the cost level of our adaptive group and the lowest level between all studied static groups was a very high 0.995. This result highlights the success of creating groups with lowered coordination costs and higher productivity.

We also studied 5 variations of the TimeRand group, again choosing values of 10, 50, 100, 200, and 500 cycles as the length of time robots waited before engaging in resolution behaviors. The dynamic TimeRand group also performed better than the static methods. Figure 2.11 displays the results from the adaptive Hill Climbing TimeRand algorithm for $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$. Again, this dynamic coordination method was able to achieve the best performance, or nearly the best, from among the various static amounts. On average, this group collected 19.2 pucks, more than the 17.6 average pucks the best static group (Timeout 50) we studied. For over half of the group sizes (18 out of 30) the dynamic group even outperformed the best static method. To confirm the statistical significance of these findings, we again performed the two tailed t-test. When comparing the dynamic timeout group to all static ones, we found p-scores of 0.0014 or less (p=0.0014 was found between the adaptive group and the Time50 method which had performed the best of the static TimeRand methods). A very high statistical correlation coefficient of 0.98 also exists between the dynamic group and the maximum productivity value taken from among all the static timeout methods over each of the 30 group sizes. Thus, we conclude that this form of adaptation is effective in raising productivity in robotic groups.

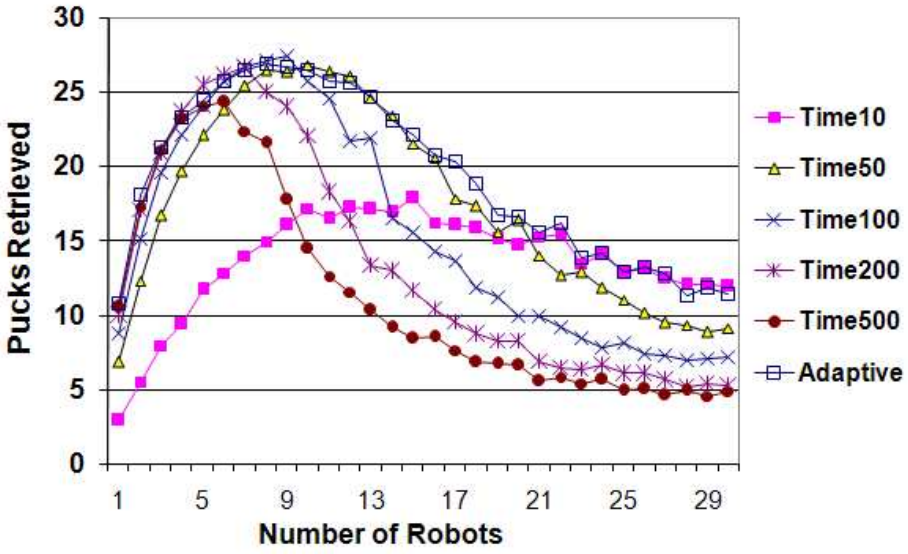

Figure 2.11: Productivity in adaptive timeout group

The second adaptation method used the value of V to switch between 3 distinct coordination methods. In the case of $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$ this involved adaptation between the Noise, Aggression, and Repel methods. The Noise group has the least costly coordination method, and was most effective in small groups up until 7 robots. At the other extreme, the Repel method fared poorly in small groups but had the best productivity in groups larger than 17 robots. For the case $\mathbf{P}_{Time} = 0$ and $\mathbf{P}_{Fuel} = 1$ this type of adaptation would involve switching between the Noise, TimeRepel, and Repel methods.

In the implementation for all adaptive methods of this type we set the values of both W_{down} and W_{up} to be one. Thus, we limited the learning portion to determining which thresholds values of V should be used to switch between methods. We again implemented versions of gradient learning and hill climbing algorithms to converge on values for these weights. Our learning algorithms converged on threshold values of V for each of the three states at 100, 200 and 300 accordingly.

Thus, if V increased by a total of 100, the robot would assume a more robust coordination method was required and would transition to use the next most robust coordination method, say from Noise to Aggression. If this method was still insufficient to resolve this instance of a projected collision, W_{up} would increase the value of V until the next threshold was reached and once again the robot would move to the next coordination method. Conversely, if that method was sufficient to resolve that incident of a projected collision, the value of W_{down} would begin to decrease the value of V and the robot could eventually move down to the next lower method of coordination.

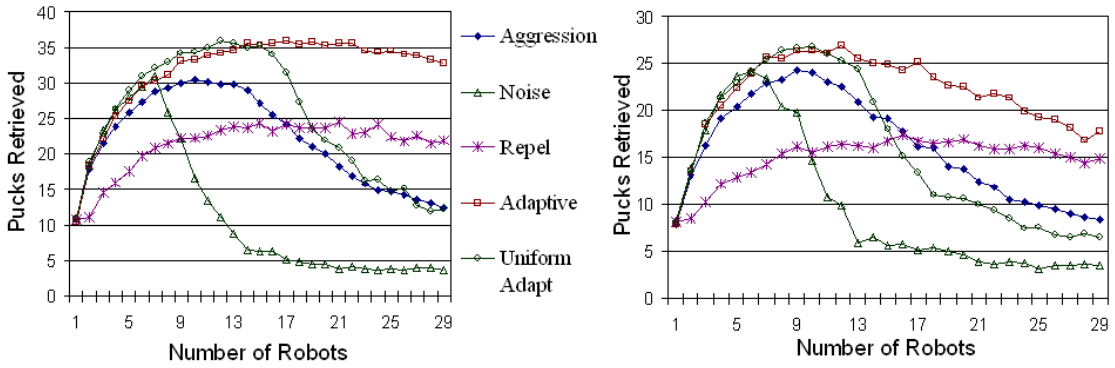

Figure 2.12: Adaptation between static groups for $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$ (on left) and adaptation between static groups for $\mathbf{P}_{Time} = 0.7$ and $\mathbf{P}_{Fuel} = 0.3$ (on right)

This second adaptive coordination heuristic was even more effective than the first approach—adaptation only within one method. Figure 2.12 contains the results from cases where cases of case of $\mathbf{P}_{Time} = 1.0$ and $\mathbf{P}_{Fuel} = 0.0$ on the right side and $\mathbf{P}_{Time} = 0.7$ and $\mathbf{P}_{Fuel} = 0.3$ on the left. In both of these cases, we graphed the productivity levels of the 3 static methods with the highest productivity as well as that of the adaptive method (learned here through Gradient Learning). The adaptive method here yielded strong productivity gains, often in excess of more than 20 percent of the static methods it was based on.

We again performed the two-tailed paired t-test on the data and found a p-value below 0.0001 between all groups and the adaptive methods, demonstrating this strong improvement.

The basic assumption of the adaptive methods we present is that all coordination acts can be done independently. Therefore, in the domains we studied, robots are able to independently choose which coordination method without impacting other team members. For example, it is possible to have one robot use the "Noise" coordination collision resolution mechanism while other robots use the "Aggression" mechanism.

However, many communication protocols exist where standardized coordination is required. To represent these situations, we also implemented an adaptive group, *Uniform Adapt* (also found in Figure 2.12). In this method, once one robot deemed it needed to switch methods, it broadcasted the method it was switching to all other robots (a global communication network was simulated) and all robots switched in turn. In order to prevent robots from quickly switching back, all robots also set their cost estimate V to the base value of this method. Potentially, this method could force certain members to use a coordination method not appropriate for its localized conditions. We hypothesized that allowing robots to autonomously adapt to their localized conditions facilitates even further productivity gains. We further develop this idea in section 2.5.3).

2.5.2 Adaptation in Multi-Robot Search

We believe the approach can be generalized to domains other than foraging. To support this claim, we implemented both adaptive methods within the search domain (previously studied in section 2.3.2.

Our first type of adaptation involves having agents adjust the strength of their coordination methods based on the needs of the domain. Again in the search domain, we demonstrate the shortcomings within static methods, and implemented the same five TimeRand variations of 10, 50, 100, 200, and 500 cycles. We then implemented an adaptive TimeRand search method using the same weight learning algorithms to set values for V_{init}, W_{up} and W_{down} as described in the previous sections. The result was a policy π which translated V to the number of cycles used when resolving any given collision event. The results of this trial for $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$ are also found in Figure 2.13. On average, we found a statistical improvement in performance in the adaptive group, with average search scores down nearly 10 percent in the adaptive group over the best levels among the static ones (TimeRand50).

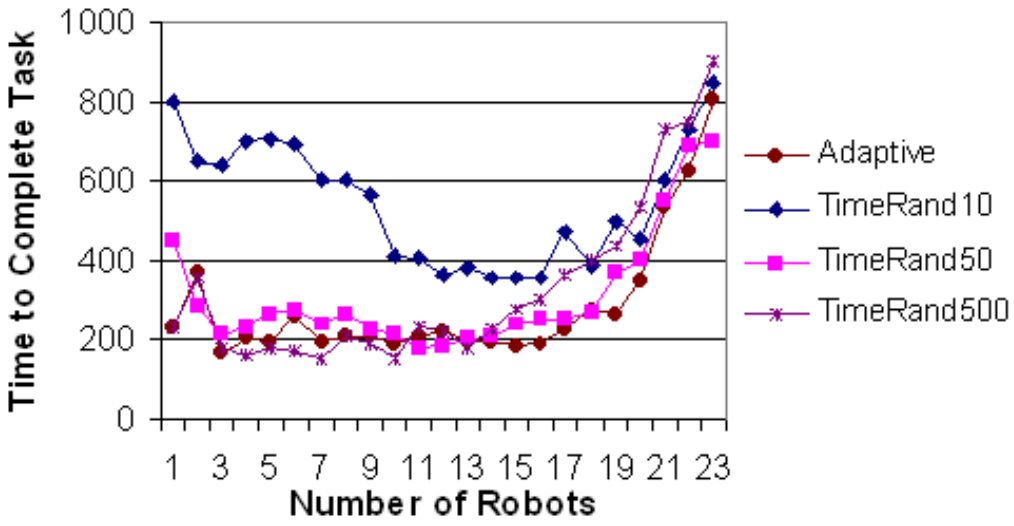

Figure 2.13: Search adaptation within TimeRand method using multi-attribute coordination costs

We were also successful in creating adaptive coordination methods that switched between the most effective coordination methods in this domain. Note that in this domain the Noise and TimeRand

were always the best two methods, regardless if the cost comprised of $\mathbf{P}_{Time} = 1$ and $\mathbf{P}_{Fuel} = 0$, $\mathbf{P}_{Time} = 0$ and $\mathbf{P}_{Fuel} = 1$, or $\mathbf{P}_{Time} = 0.7$ and $\mathbf{P}_{Fuel} = 0.3$. We used the same methodology to create an adaptive search method with each robot using the CCC cost estimate V to effectively switch between these methods.

Figure 2.14 shows the Noise, TimeRand and Adaptive groups in the instance of $\mathbf{P}_{Time} = 0.7$ and $\mathbf{P}_{Fuel} = 0.3$. On the left side, we denote the productivity graphs with the X-axis represents the size of the group, and the Y-axis displaying the search time, measured in seconds, until that group completed its task. On the right side, we display the CCC measures for these groups, with the Y-axis displaying the normalized CCC measure weighted between time and fuel (normalized out of 250 units). In order to establish the statistical significance of the results we performed the two-tailed paired t-test between the adaptive methods and the static ones they were based on. All results were below the 0.05 confidence level (between 0.01 and 0.04 in all three groups).

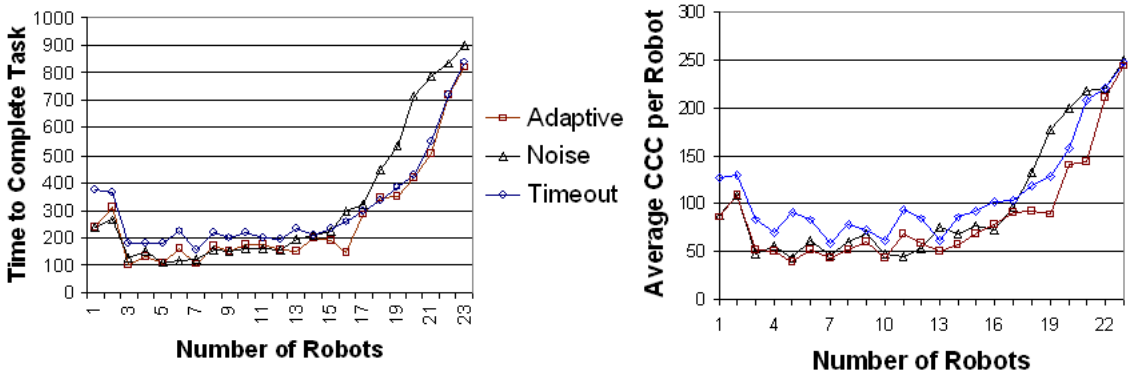

Figure 2.14: Search adaptation using multi-attribute coordination costs

2.5.3 Quickly and Significantly Improving Performance

We found that some flexibility exists in setting the weights: V_{init}, W_{up}, and W_{down}. Our results demonstrate that even results that were far

from optimal were still a significantly improvement from the static methods they were based on. This is because a value of V_{init} being initially set too high was soon corrected by the weights in W_{down}. Conversely an initial value set too low can be quickly rectified by the weights in W_{up}. Figure 2.15 depicts the productivity of three adaptive repel foraging groups with values for V_{init} of 300, 450 and 600 and identical values for W_{up} and W_{down}. Note that while differences exist, for most group sizes these differences were not statically significant.

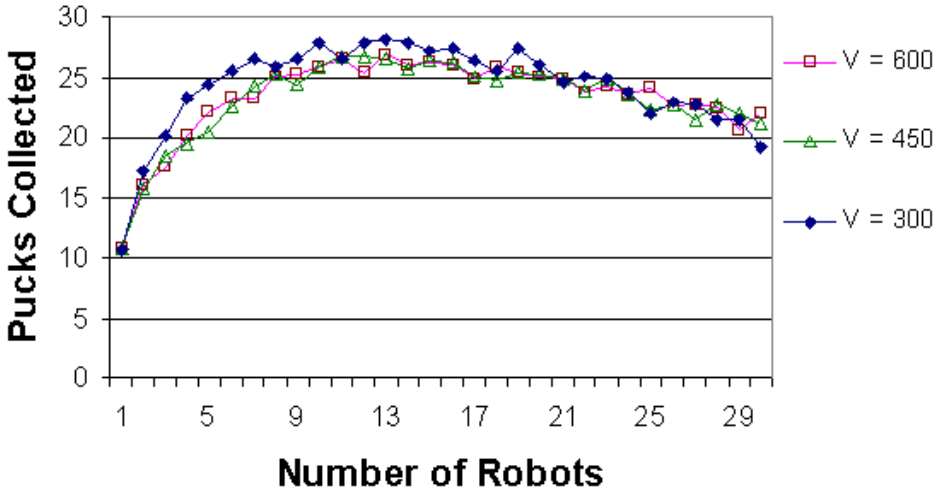

Figure 2.15: Three adaptive repel groups with different values for V_{init}

Figure 2.16 demonstrates the success of the weighted heuristic approach with only minimal learning. This graph represents three iterations in the gradient learning implementation for the adaptive foraging repel method. Our initial policy was based on Repel200, which on average had the highest average productivity over the 1–30 robot interval. In the first adaptive iteration (Gradient1) we used a value of 200 for V_{init} and naive values of 10 for both W_{up} and W_{down}. In subsequent trials (Gradient2, Gradient3), gradient learning was used to tweak these naive values. Two issues are noteworthy in this graph: First, recall that in the first evaluation method, the policy π is eval-

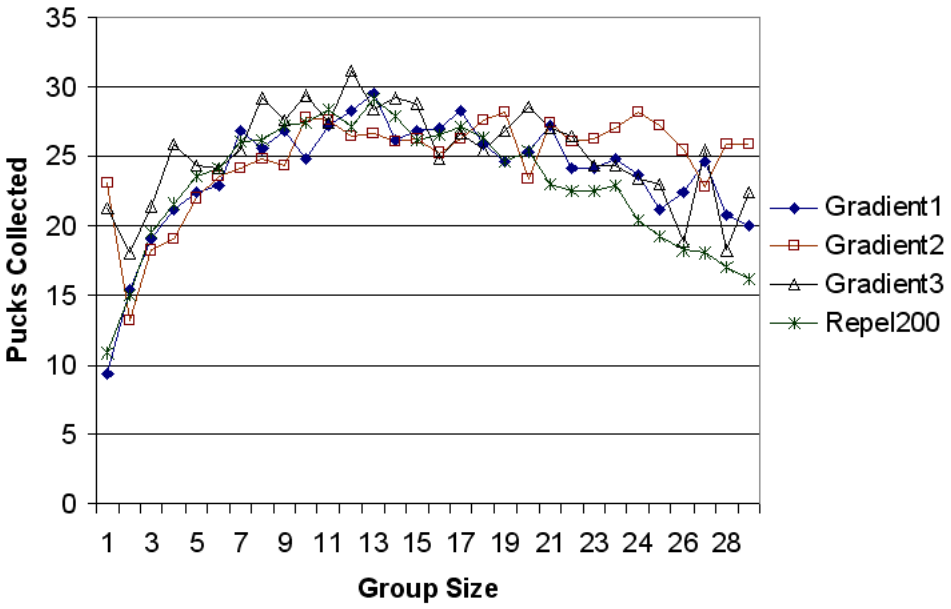

Figure 2.16: Three iterations of the adaptive repelling groups using gradient learning

uated from averaging five trials over the entire group range. Notice the large variance between trials. This illustrates the difficulty in learning an optimal weight value without extensive trials. Second, note that despite this difficulty, gradient learning quickly improved the weights used in the algorithms. Even within the first iteration (Gradient2) the adaptive group averaged approximately 5% improved performance, while by only the third iteration a near local optimum was achieved with an average performance increase of 10%.

2.5.4 Large Productivity Gains

Not only does coordination adaptation based on CCC estimates yield productivity gains after short learning periods, but these productivity gains are often quite large—beyond any of the static methods they are based on. For example, we previously presented two types of foraging adaptive groups, *Adaptive* and *Uniform Adapt* that often significantly exceeded the productivity levels of the methods they were based on.

At first glance, this result is surprising. One would assume adaptation is only capable of achieving results in line with the best levels of productivity for the methods it was based on, not significantly higher.

We claim that the root of these productivity gains is the ability of these methods to switch between coordination methods as dictated by fluctuating domain conditions. Thus, during the course of one trial, one robot may switch between its Noise, Aggression, and Repel coordination methods many times. Our goal is not to converge on any one coordination method, as that method can often change as the possible of collisions grows or dissipates. To demonstrate this point, we studied the average CCC estimate, V, each robot within the various group sizes contained. Recall that this value ranged from 0–300 with values of 0–100 mapped to the Noise method, values of between 100 and 200 mapped to the Aggression method, and larger values to the Repel method. Assuming the goal was to converge on the one static method with the highest productivity, one would assume these robots would have average values of V of over 200 in groups larger than 17 (where the static Repel group fared best). However, as Figure 2.17 demonstrates, this was not the case, and average values for V ranged between 0 and 200 regardless of the Adaptive group's size. This result implies that even in large groups, robots did not use the most expensive method (Repel) for large portions of the trials. For example, in one foraging trial of 25 robots using the *Adaptive*, method, the entire team spent on average 56 percent of their time in the Noise behavior, 11 percent in Aggression behavior, and 33 percent in the Repel behavior. Thus, the average value of V never rose above 200 because the group never spent a majority of their time using the most costly coordination methods.

Our working hypothesis is that fluctuations in the level of collisions

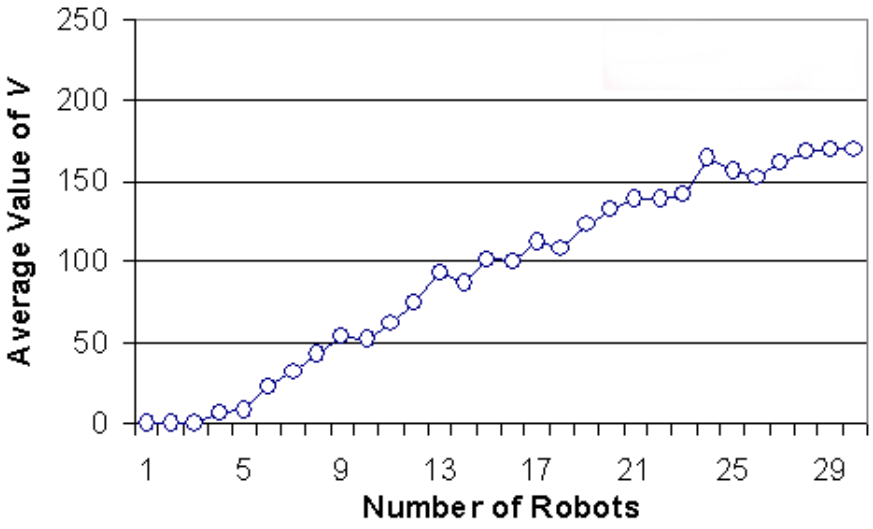

Figure 2.17: Average threshold values, V, between robots using adaptive coordination method when $\mathbf{P}_{Time} = 1.0$ and $\mathbf{P}_{Fuel} = 0.0$

even within one trial allow for this adaptive method to outperform the static ones it is based on. The *Adaptive* method adapted to these fluctuations, yielding the marked improvement in this group's productivity over other groups. As empirical evidence of these fluctuation within trials, Figure 2.18 represents the percentage of robots that are colliding throughout the course of three trials (540000 cycles) in groups of 25 robots. The X-axis in this graph represents the number of cycles elapsed in the trial (measured in hundreds of cycles), while the Y-axis measures the percentage of robots colliding at that time. We found that these values do in fact fluctuate, at times sharply, throughout almost all foraging trials. This further illustrates the danger in attempting to converge on one ideal coordination method, even within one trial.

We believe this is also the reason why the Adaptive method significantly outperformed the *Uniform Adapt* group in larger group sizes. At times, the Uniform Adapt approach may be advantageous as some robots could cue others as to the best coordination method to use.

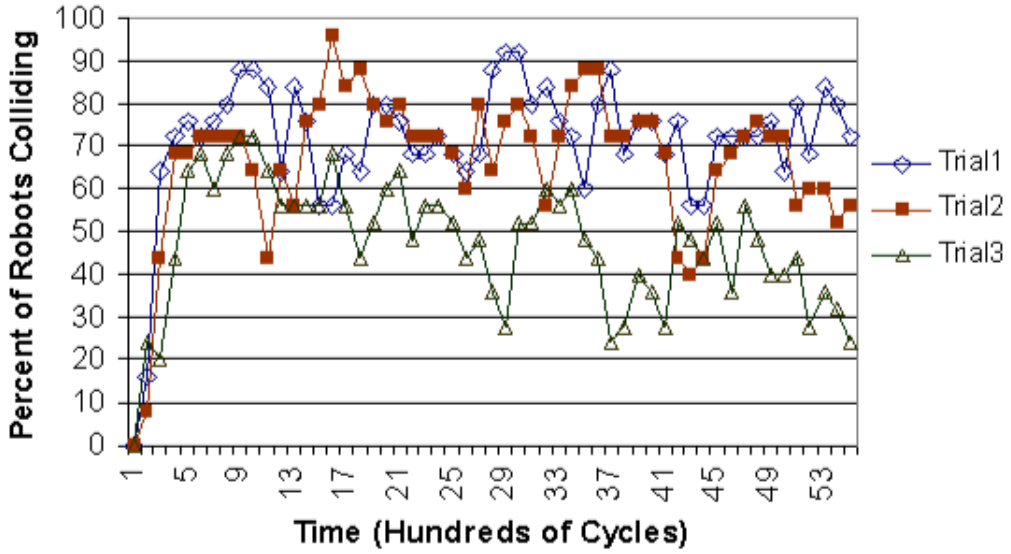

Figure 2.18: Fluctuations in collisions over time

Notice how this group did have slightly higher productivity in small to medium groups. However, we believe the Uniform Adapt method has two major drawbacks. First, it requires communication between robots, a factor that would likely add another coordination cost, $\mathcal{C}_i^j$ to every agent in a group. However, even beyond this point, we believe the first approach is more effective in allowing robots to adapt to their local domain conditions. In domains with dynamics, such as the ones we studied, at least one robot is typically not colliding, and thus would naturally choose the least costly Noise coordination method. In the Uniform method, this one robot could force the entire group to switch back to this method, accounting for the lower productivity in this group when more costly methods were justified. In the future, we hope to further study how adaptation can yield improvements in productivity, even when standardized adaptation is required.

Finally, observe that the gains from the Adaptive approach in the foraging domain that switched between coordination methods (see Figure 2.12) were much greater than the adaptive methods that tweaked

the parameter strength within one method (Figure 2.10). We believe this difference is primarily due to the large differences in the density distributions and cost functions (refer to Figures 2.3 and 2.5) created by these methods in this domain. As a result, when the Adaptive approach switched between these sharply different coordination methods it benefited from larger productivity gains.

In contrast, the first type of adaptation, i.e. adaptation within one coordination method, did not have as large differences in the variations within one coordination method (see Figure 2.4). As a result, adaptation did not facilitate radically different approaches to coordination, and productivity gains from this category of adaptation did not significantly outperform the methods it was based on. Similarly, the search domain only had two methods to switch between, with only modest differences in their cost functions (see Figure 2.14). We believe that this prevented the adaptive methods in this domain from realizing even larger productivity improvements.

2.6 Conclusion and Future Work

In this paper we argue that the coordination cost a single robot generates is a primary factor in determining the productivity of the entire group. In theory, robots should consistently demonstrate increasing marginal productivity increases. However, limiting production resources, such as the spatial limitations inherent in many robotic groups, prevent productivity gains by this theoretical amount. At times, adding robots then hurts performance, as was previously noted [66, 71]. We present a model for evaluating multi-attribute coordination cost functions that a single robot contains. Our CCC (combined coordination cost) measure quantifies a weighted sum of all produc-

tion resource conflicts between members of a group. While other team measurements are possible, we found that focusing on this cost alone facilitates effective comparison between different coordination methods. Our approach requires no centralized mechanism, with accurate coordination measures being taken autonomously by members of the group. We present two adaptive coordination methods based on our measurement which both improve the group's performance and scalability properties in a statistically significant fashion in the foraging and search robotic domains we studied.

For future work, several directions are possible. We believe it may be possible to use the coordination measurements to predict when adding an agent to the group will be helpful. Team sizes could thus be modified to maximize the use of production resources. We also hope to study if similar measurements could model gains each robot adds to its group. Such a measurement would be useful for purposes of task allocation as it could identify which team member is best suited to perform given tasks. We are hopeful that the use of the CCC measure will replace domain and task specific cost functions. We believe this approach could facilitate additional advances in agent and robotic team research.

Chapter 3

Adaptive Robotic Communication Using Coordination Costs

3.1 Introduction

In the previous chapter, we assumed the robots could choose their coordination methods independently of the other team members' actions. This was because the robots coordination methods were inherently compatible and they did not need any feedback from other team members. Therefore, all of the adaptive methods presented, with the notable except of the **Uniform Adapt** method, allowed robots to freely choose the best coordination method without concern of the impact this choice. For example, it was possible to have one robot use the **Noise** coordination collision resolution mechanism without impacting other robots which used the **Aggression** mechanism. As a result, adaptation was possible without any communication or standardization between mechanisms.

However, many coordination methods do require some type of coordination synchronization between methods. Most notably, communication typically requires this type of standardization. Picture, for example, one person attempting a conversation in Dutch with some-

one who only knows English. We found that adaptive approaches were less effective when synchronization was required between methods, as was the case in the **Uniform Adapt** method in the last chapter.

In theory, communication should not be a disadvantage–the more information a robot has, the better. However, assuming communication has a cost, one must also consider the resources consumed in communication, and whether the cost of communication appropriately matches the needs of that. Furthermore, effective methods of standardization must be addressed. Towards this goal, recent study has addressed questions such as what to communicate and to whom [30, 67, 70]. We believe that different communication schemes are best suited for different environmental conditions. Because no one communication method is always most effective, one way to improve the use of communications in coordination is to find a mechanism for switching between different communication protocols so as to match the given environment.

In this chapter we address the challenging question of how to create effective adaptive communication frameworks, even when synchronized coordination is required. Our solution uses a coordination cost measure that quantifies all resources spent on coordination activities. Our model explicitly includes resources such as the time and energy spent communicating. In situations where conflicts between group members are common, more robust means of communication, such as centralized models, are most effective. When collisions are rare, coordination methods that do not communicate and thus have the lowest overhead, work best.

We present two novel domain-independent adaptive communication methods that use communication cost estimates to alter their communication approach based on domain conditions. In our first approach,

robots uniformly switch their communication scheme between differing communication approaches. In this method, robots contain full implementations of several communication methods, and switch between them as needed. In contrast, our second approach represents a generalized communication scheme, that allows each robot to adapt independently to its domain conditions. Each robot creates its own communication range radius (which we refer to as its *neighborhood of communication*), to create a sliding scale of communication between localized to centralized methods. Each robot uses its coordination cost estimate to determine how large its neighborhood should be.

To evaluate these adaptive methods, we performed thousands of trials using an established robotic simulator in a multi-robot foraging task. We tested groups of varying sizes and communication methods. We found that groups that used the adaptive methods often significantly exceeded the best productivity levels of the non-adaptive algorithms they were based on.

3.2 Existing Communication Schemes

A major challenge to designers of robotic groups exists in choosing an optimal communication method. Many practical frameworks have been presented for use within robotic teams [13, 16, 21, 27, 30, 34, 51, 70] and can generally be assigned to categories of no communication, localized, and centralized approaches.

It is possible to create effective group behavior without any communication [6]. For example, the Stigmergy concept [27] involves group members basing their actions by observing how other group members previously modified their environment. This approach has been shown to be effective in several animal and robotic domains [27] without us-

ing any explicit communication. Coordination without communication can potentially facilitate better adaptability, robustness and scalability qualities over methods using communication [67]. Additionally, the lack of communication also allows such methods to be implemented on simpler robots. However, such algorithms often require powerful and accurate sensing capabilities [51]. Also, our results demonstrate that groups implementing these methods did not always provide the highest levels of productivity, especially within dynamic domains where frequent coordination conflicts exist.

A second set of approaches attempt to improve group performance by having robots locally communicate information [30, 51]. For example, work of Jäger and Nebel [30] present a method whereby robots nearing a collision stopped to exchange trajectory information. They then successfully detect and resolve deadlock conditions of two or more robots mutually blocking. However, their trajectory planning method was not able to perform well in groups of over five robots. In contrast, Mataric [51] reported that a local communication scheme scaled well with group size. One key difference seems to lie within the local communication implementations. In Jäger's algorithm, one or more robots must stop moving during trajectory replanning. We believe this led to a breakdown in the system once the group size grew. Mataric's locally communicating robots broadcast information while continuing their foraging task. This allowed for better scalability qualities.

A third type of approach involves the use of some type of central repository of information [70]. This centralized body, which could also be implemented as one "expert" teammate, would then be able to easily share its store of pooled information with other teammates. While this approach allows for free information sharing and can thus improve performance, several drawbacks are evident. First, the centralized

mechanism creates a single point of failure. The cost of communication is also likely to be large, and requires hardware and bandwidth suitable for simultaneous communication with the centralized body. While these drawbacks are at times significant, they may be justified given the needs of the domain.

In this chapter, we assume that representative communication methods from these categories are predefined, and have been implemented with optimal values for their exact parameters given domain conditions. Several approaches exist for finding these parameters within a given coordination method. For example, work by Yoshida et al. [16] presented a framework to derive an optimal localized communication area betweens within groups of robots to share information in a minimum of time. This approach assumes domain conditions such as spatial distributions and the probability of information transmission can be readily calculated. Previously, Goldberg and Mataric [21] focused on *interference* (which they defined as the time robots spent colliding) as a basis for measuring a coordination method's effectiveness. However, they did not address how to create adaptive methods based on interference. Our last chapter describes work built upon this interference definition to include all resources spent resolving coordination conflicts including the time spent before and after collisions. We then demonstrated that parameter tweaking is possible through this measure. The advantage to this approach over the work of Yoshida et al. [16] is its ability to allow robots to autonomously adapt, even in dynamic environments. However, in contrast to their work, our previous chapter did not address how to effectively address communication adaptation.

In this chapter, we use coordination cost measures to compare a given set of communication methods and to create adaptive methods

based on these existing methods. We explicitly model all resources spent on coordination activities including the resources spent on communication even if they do not detract from the time to complete the task. Our goal was to properly match communication methods to domain conditions, while considering their relative costs. Furthermore, adaptation between communication schemes presents new challenges, since many protocols require standardized communication between all team members. These challenges are addressed in this work.

3.3 Using Coordination Costs to Adapt Communications

In the last chapter, we presented a combined coordination cost measure, CCC, to quantify the effectiveness of a coordination method. We modeled every robot's coordination cost $\mathcal{C}_i$, as a factor that impacts the entire group's productivity. In this chapter, we extend this measure to analyze two cost categories: (i) costs relating to communication and (ii) proactive and/or reactive collision resolution behaviors. We focus on the time and energy spent communicating and in the consequent resolutions behaviors (see Implementation Section for full details). We then combine these factors to create a multi-attribute cost function based on the Simple Additive Weighting (SAW) method [77] often used for multi-attribute utility functions. While methods with no communication have no $\mathcal{C}_i$ for communication, this method could not always successfully resolve collisions and then spent more resources on collision resolution behaviors, or another $\mathcal{C}_i$. Conversely, centralized methods incurred a communication cost $\mathcal{C}_i$ that often eclipsed the needs of the domain and weighed heavily on productivity. Other communication issues, such as bandwidth limitations, can similarly

be categorized as additional cost factors as they impact any specific robot. For example, if a robot needed to retransmit a message due to limited shared bandwidth, costs in terms of additional time latency and energy used in retransmission are likely to result.

We use this extended CCC measure for online adaptation between communication schemes. We present two types of adaptive methods: (i) uniform communication adaptation (ii) adaptive neighborhoods of communication. Both methods led to significant increases in productivity over static approaches (see Experiments section).

3.3.1 Uniform Switching Between Methods

In our first method, all robots simultaneously switch between mutually exclusive communication methods as needed. In order to facilitate this form of adaptation, each robot autonomously maintains a cost estimate, V used to decide which communication method to use. As a robot detects no resource conflicts, it decreases an estimate of this cost, V, by an amount W_{down}. When a robot senses a conflict is occurring, the value of V is increased by an amount W_{up}. The values for V are then mapped to a set of communication schemes methods ranging from those with little cost overhead such as those with no communication, to more robust methods with higher overheads such as the localized and centralized methods. As the level of projected conflicts rises (as becomes more likely in larger group sizes) the value of V rises in turn, and the robots use progressively more aggressive communication methods to more effectively resolve projected collisions. While these activities themselves constitute a cost that detracts from the group's productivity, they are necessary as more simple behaviors did not suffice. As different coordination methods often have different

costs, $\mathcal{C}_i$ for a given domain, we believed this approach could be used to significantly improve the productivity of the group.

Several key issues needed to be addressed in implementing this method with groups of robots. First, we assumed that all group members are aware of the overheads associated with various coordination methods, and can order them based on their relative complexities. This ordering can be derived from theoretical analysis or through observation (as we do in later in this paper). Second, an approach to quickly set the weights, W_{up}, and W_{down} used within our algorithms is needed. Robotic domains often contain dynamics that render a learned policy obsolete very quickly. Thus, our approach is to sacrifice finding a globally optimal policy in exchange for finding a locally optimal policy after a much shorter training period for our weights. In this chapter, we used a gradient learning procedure to achieve this result.

Next, it must be noted that uniform adaptation requires all robots to change communication in sync because of the mutual exclusivity of the methods used. For example, it is impossible for one robot to use a centralized method, with others using one without communication, as the centralized approach is based on information from all team members. As a result, once any one robot in the group autonomously decided it needed to switch communication schemes, a communication change must also occur within all other team members. This could force certain members to use a more expensive communication method than it locally found necessary. We relaxed these requirements in the second adaptive method, presented in the next section.

Finally, care must be taken to prevent the robots from quickly oscillating between methods based on their localized conditions. In our implementation, communication adaptation was triggered once one ro-

bot's value for V exceeded a certain threshold. After this point, that robot broadcasted which method it was switching to and all group members would change in kind and reinitialize their cost estimates V to this new value. Furthermore, we also used domain specific information, such as prioritizing collisions closer to the home base within our foraging domain. In this fashion, we partially limited the types of triggers to those of importance to the entire group. Once again, our second type of communication adaptation relaxes this requirement and is effective without any such heuristics.

3.3.2 Adaptive Neighborhoods of Communication

The advantage in our first adaptive approach lies in its simplicity. Our uniform adaptive approach switches between existing coordination methods based on estimated coordination cost. Assuming one analyzes a new domain with completely different communication methods, and can order the communication methods based on their communication costs, this approach will be equally valid as it implements existing methods and reaches the highest levels of productivity from among those methods–whatever they may be.

In contrast, our second adaptation method is a parameterized generalization of the three specific categories of communication methods (No-Communication, Localized, and Centralized). As many robotic domains use elements of these same methods [13, 16, 27, 30, 34, 70], we reason that a similar approach is likely to work in these and other domains as well.

The basis of this approach is introducing a parameter to control how large a radius of communication is used by each robot. This method uses a distance d inside which robots exchange information, which

we term its communication neighborhood. Formally, this radius of communication could be considered a neighborhood Γ of size d, created from robot v and includes all teammates, u, inside this radius. As such, we represent the neighborhood as $\Gamma_d(v) = \{u|\ u \text{ robot}, dist(u, v) \leq d\}$.

Adjusting the value of d in Γ_d can be used to approximate the previously studied communication categories. Assuming d is set to zero, no communication will ever be exchanged and this method is trivially equivalent to the No-Communication method. Assuming d is set to some small amount, ε, this method will become similar to the Localized method and information will be exchanged only with the robot it is about to collide with. If d is set to the radius of the domain, the neighborhood of communication encompasses all teammates this method becomes similar to the Centralized method. Thus, the degree of centralization exclusively depends on the value of d.

3.4 Implementation Details

We again used the Teambots [5] simulator to implement communication schemes involving no communication, localized and centralized approaches within groups of Nomad N150 foraging robots. In these experiments, there were a total of 60 target pucks spread throughout an operating area of approximately 10 by 10 meters. We measured how many pucks were delivered to the goal region within 9 minutes by groups of 2–30 robots using each communication type. We averaged the results of 100 trials for each group size with the robots being placed at random initial positions for each run.

We created experiment sets measuring the time and energy spent in two coordination categories–communication and collision resolution.

The coordination costs in our first set of experiments involved the time spent in communication and collision resolution behaviors out of each trial's total time of 9 minutes. In our second set of experiments, we allocated each robot 500 units of fuel. We assumed most of the fuel was used by the robots to move, with a smaller amount (1 unit per 100 seconds) used to maintain basic sensors and processing. For the time based experiments, we assumed robots pairs stopped for 1/5 of a second to communicate, representing some methods [30] where robots stop to exchange information. In the energy based localized experiments, we assumed robots did not stop to communicate, as is the case with other methods [51], but each robot still spent 0.3 units of fuel per communication exchange. Our coordination cost involved the amount of fuel that was used in communication and repulsion behaviors.

All three communication schemes were similar in that they resolved collisions by mutually repelling once they sensed a teammate within a certain safe distance ε, which we set to 1.5 robot radii. Once within this distance, robots acted as they were in danger of colliding and used repulsions schemes to resolve their collision(s). The No-Communication method was unique in that robots never used time or fuel to communicate, and thus only had costs relating to the repulsion behaviors robots engaged in. However, this method assumed domain specific information, namely it used the robot's autonomously computed scalar distance, $\mathcal{S}$, from its location to the home base in the domain. Robots used a function of this distance, which we implemented to be 5 times $\mathcal{S}$ and rounded to the closest second, as the time to repel from its teammate(s) after a projected collision.

The localized method used less domain specific information and is similar to the localized methods previously proposed [30, 51]. Com-

munication between robots was initiated once it was in danger of colliding–a teammate came within the ε distance. After this event, these group members would exchange information above their trajectories (here their relative distances from their typical target, their home base). The closer robot then moved forward, while the other robot repelled for a fixed period of 20 seconds.

The final method, *Centralized*, used a centralized server with a database of the location of all the robots similar to other centralized methods [70]. Within this method, one of two events triggered communication. First, as with the localized method, robots dropping within the ε distance initiated communication by reporting its position, done here with the centralized server. The server then reported back a repel value based on its relative position to all other teammates. However, in order for the server to store a good estimate of the positions of all robots, a second, often more frequent type of communication was needed where each robot reported its position to the server with frequency L. If this communication occurred too frequently, this central database would have the best estimate of positions, but the time or energy spent on communication would spike, and productivity would plummet. If communication was infrequent, the latency of the information stored on the server would create outdated data. This in turn would reduce the effectiveness of this method, and result in more collisions.

It is important to stress that the focus of this work is selecting the best communication method from a known set of options, and not to find optimal parameters within any one given communication method. We refer the reader to previous work [16, 64] on how to theoretically or empirically derive parameters within one communication method. This work is based on the understanding that a high negative correlation exists between each groups' productivity and our coordination

cost, regardless of the exact implementation for the parameters used in the No-Communication, Localized and Centralized methods.

While we consider the neighborhood communication approach to be a parameterized generalization of the three previously described categories, some implementation details differ in this method over the static ones it emulates. Within this method, once any robot A, detects another robot within the ε distance, it initiates communication with all robots found within the $\Gamma_d(A)$ area. All robots in $\Gamma_d(A)$ must then report back to Robot A with their projected trajectories. Robot A then sorts all robots' trajectories by their relative distances from the home base in the domain. This robot then reports back to every robot within $\Gamma_d(A)$ a repel value based on that robot's relative position in the neighborhood. All robots, including the initiating robot (robot A), then accept this value and immediately engage in repel behaviors for the dictated length of time. It is possible that a robot may be a member of more than one neighborhood. In such cases, robots accept the larger repel value regardless of the sender.

While the repel amounts of the robot initiating communication (Robot A) are calculated in a similar fashion to the previously described centralized method, here these values are calculated by members of the team, instead of one centralized server. The radius of communication in the centralized approach is the full width of the domain, while the Γ_d radius is typically much smaller. However, the biggest difference in implementing this approach is how repel values are obtained. Robots in previous methods only repelled based on communication received after dropping within the ε distance. In this method, robots may repel if they enter the Γ_d radius even if they are not in immediate danger of colliding. The reason for this is as follows. As robots within the Γ_d radius are typically close to each other, we found that these ro-

bots often would soon initiate their own radii of communication. In other methods this was not a concern, as other teammates were not effected by this phenomenon. However, here this would create multiple neighborhoods involving the same teammates. Thus, proactively assigning repel values was crucial for containing communication costs as Γ_d grew.

3.5 Experimental Results

The first set of experiments attempts to first lend support to the underlying hypothesis, that the combined coordination cost measure is in fact correlated to the productivity of the different groups. Our results from experiments involving time and energy costs support the claim that the best method of communication does change with domain conditions. Figure 3.1 contains the results from the time based coordination cost trials. In the top portion of the graph, the X-axis represents the group size, and the Y-axis the number of pucks successfully retrieved within each group. The No-Communication approach worked best in small groups where collisions were less likely. In medium sized groups, the localized approach worked better. As collisions became frequent, the large amount of communication inherent in the centralized method became justified, and this group performed significantly better. The total cost of coordination as a function of time are presented in the lower graph in Figure 3.1.

Notice that the No-Communication method was only effective in minimizing this cost (presented as the Y-axis and measured in seconds) for small groups (the X-axis). In larger groups, this method engaged in more repulsion behaviors because it was not successful in collision resolution without communication. The localized group maintained

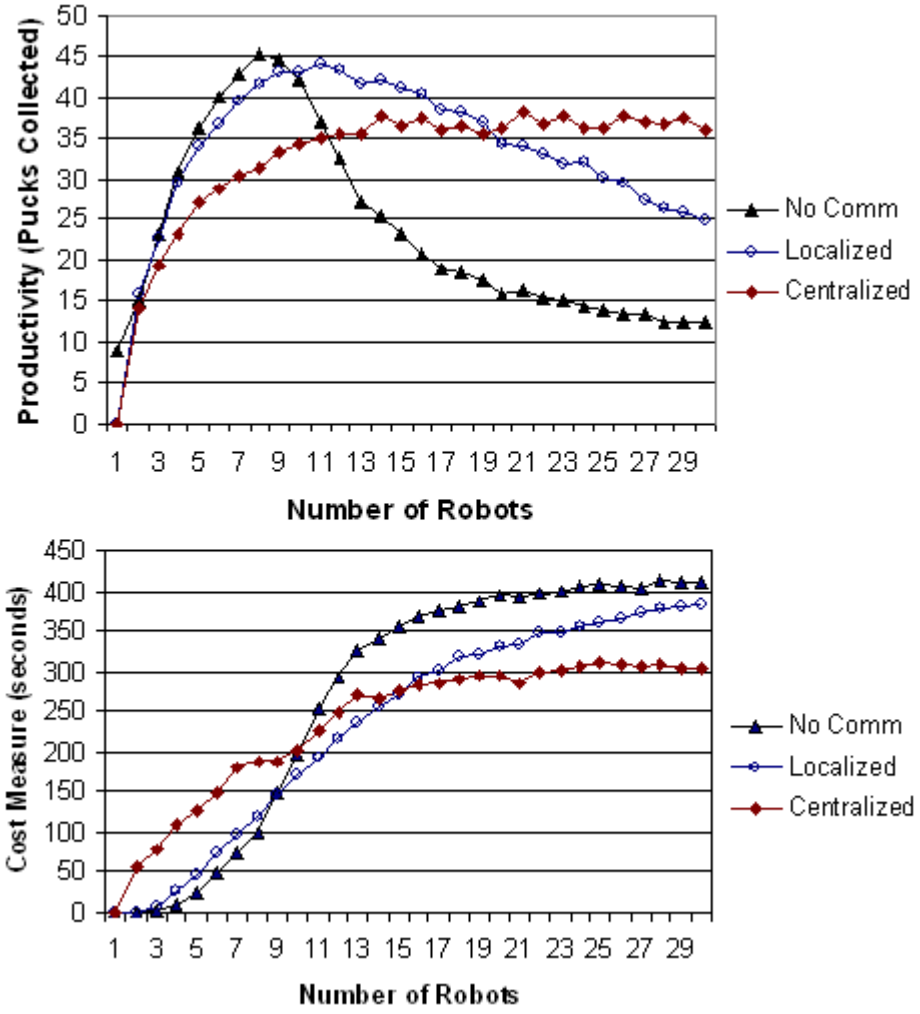

Figure 3.1: Comparing levels of time spent on communication in different group sizes. Results averaged from 100 trials per datapoint.

near linear levels of its coordination cost with respect to the group size but the communication costs within this group made it less effective in smaller groups. The centralized method had the largest cost overhead, but these costs were not as effected by group size. As a result, this group achieved the highest productivity in large groups.

We also found a very strong negative correlation between the coordination cost based on *energy*, and the groups' corresponding productivity. In these trials, we measured the total energy used by our groups in coordination behaviors, including communication. As was

the case in the time based experiments, we again found the best method changed as the group size increased, and thus collisions became more likely. The No-Communication method again fared best in small groups, the localized one in medium groups with the centralized method faring best in larger groups.

Both sets of experiments had similar results in that the team's productivity was strongly negatively correlated with coordination costs. In the time experiments, we found an average correlation of -0.96 between the productivity found in groups of 2–30 robots and the group's corresponding cost. In the equivalent energy based experiments, we found a value of -0.95.

It is important to stress that we implemented several variations of the parameters used in the No-Communication, Localized and Centralized methods with all variations also demonstrating this same high negative correlation as well. The parameters used within these methods affected the coordination cost, and thus the productivity outcome. For example, we studied 7 latency variations within the Centralized method in both experiment sets. These groups enforced maximal latency periods of L set to 0.1, 0.3, 1, 5, 10, 30 and 60 seconds. In the time based experiments we found that a latency of 1 seconds often yielded average productivity level near 45 pucks. In the energy based experiments, a latency of 1 or 5 seconds yielded similar results of an average productivity of less than 35 pucks (see figure 3.2 below). This difference occurred because the cost of communication (1/10 of a second) in the first trials was different than this cost (0.3 units of fuel) in the second. However, in both cases the productivity of these variations was highly negatively correlated with their relative coordination costs. In the first case, we found a correlation of -0.95 between these latency variations and the corresponding coordination cost based on

time. In the trials based on fuel, this value was -0.97.

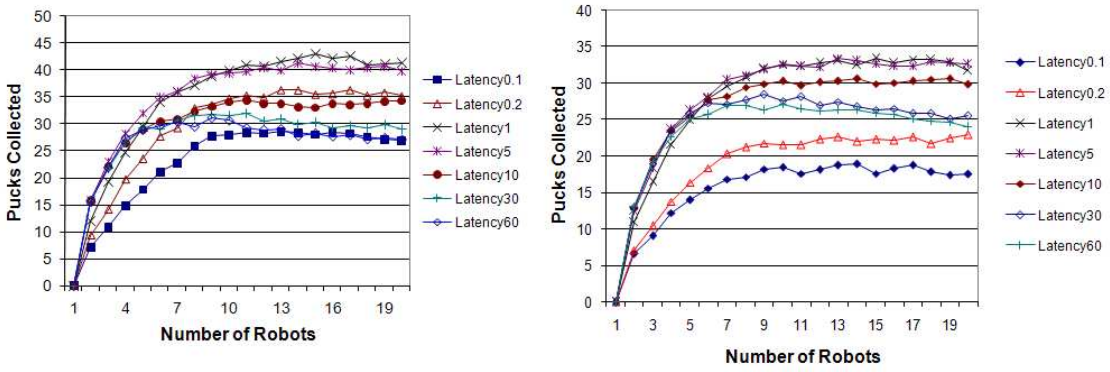

Figure 3.2: Comparing latency differences and productivity levels for centralized method in time (left) and energy (right) experiments. Results averaged from 100 trials per datapoint.

Within both experiments we found that latencies set too high typically converged with those groups where it was set too short. For example, figure 3.2 displays our latency productivity variations in the time (displayed on the left) and energy trial sets (on right). We graphed the productivity levels (Y-axis) of the 7 latency variations as a function of the group size (X-axis). Notice how methods that update their information frequently often have the same productivity levels of methods that infrequently communicate. For example, in the time experiments, Latency0.1 (communication every 0.2 seconds) converged with Latency60 (communication one a minute). While Latency0.1's frequent communication had its cost primarily due to communication, Latency60's infrequent communication often made the database of teammates' positions inaccurate. An attempt to unwisely reduce communication, and this type of cost, led to an increase of repulsion behaviors, or a second type coordination cost.

Similarly, we found that no one neighborhood size always fared best. We compared the productivity levels of foraging groups where d was set to 1, 2, 3, 5 and 50 robot lengths. Recall that ε is approximately

1 robot length (1.5 radii). Thus Γ_1 represents the nearly localized variation with Γ_{50} corresponding to the nearly centralized version of this method.

Figure 3.3 represents the relative productivity levels for these static neighborhood groups relative to the energy costs levels measured in these groups. Notice how in small groups, Γ_1 yielded the highest average productivity. As we have seen, when possible, resources spent on coordination, here by creating large communication neighborhoods, should be avoided. As small areas of communication sufficed in small groups, this approach had the highest productivity. As the group size grew, additional communication was necessary to maintain high productivity levels. As a result, larger neighborhoods were necessary and groups with Γ_5 resulted in the highest productivity. However, forcing too much communication when not necessary created communication costs that reduced productivity to levels found in methods that spend too few resources on communication. In this method, the productivity level of the Γ_{50} method, which created too large a neighborhood, approached those of Γ_1, which did not create a large enough one. We again found a strong correlation between the various Γ_d variations and the groups' corresponding coordination costs and productivity with an average negative correlation of -0.96.

Based on the confirmed hypothesis, that the cost measure is indeed correlated (negatively) with performance, the next set of experiments evaluated the performance of the two adaptive methods compared to the static methods on which they were based. Figure 3.4 shows the results from these experiments. Notice that both adaptive approaches approximated or significantly exceeded the highest productivity levels of the static methods (No Communication, Local, and Centralized methods) they were based on, especially in medium to large groups.

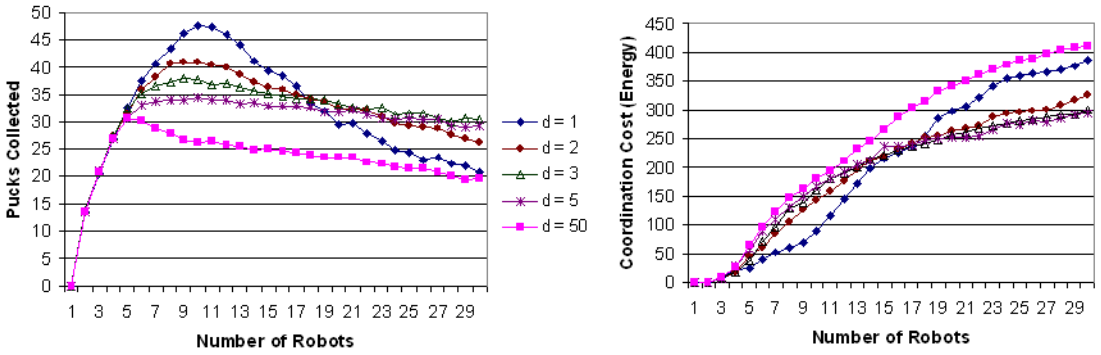

Figure 3.3: The impact of varying neighborhood sizes (d) on productivity levels and costs in energy experiments. Results averaged from 100 trials per datapoint.

We attribute the success of both methods to their ability to change communication methods to the needs of the domain. We believe that the neighborhood method outperformed the uniform one as it was allowed to create locally different neighborhood sizes, something none of the static neighborhood methods were capable of. This in turn facilitated better adaptation and higher productivity.

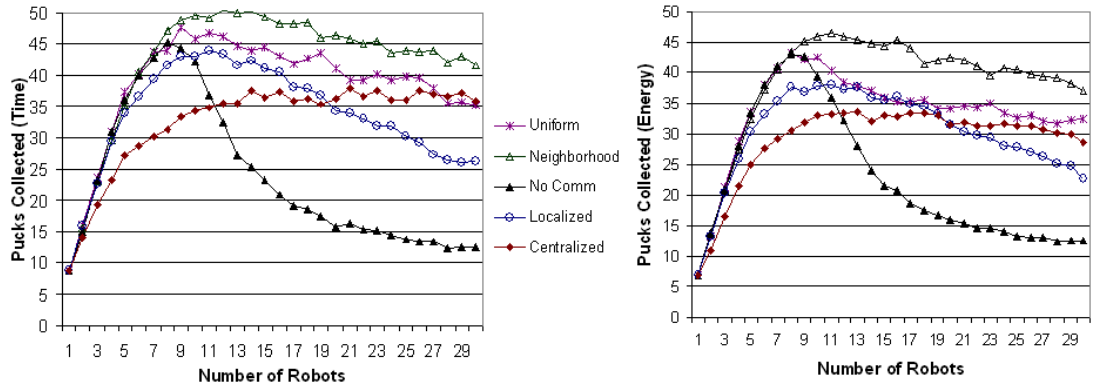

Figure 3.4: Comparing adaptive communication methods based on time and energy costs to static methods. Results averaged from 100 trials per datapoint.

To evaluate the statistical significance of these results, we conducted the two tailed t-test and a 1-factor ANOVA test comparing our adaptive groups and the three static groups they were based on. In all cases, in both time and energy categories, the null hypothesis p values

were below 0.001. This confirms the hypothesis that we can improve productivity through creating adaptive methods based on communication costs.

3.6 Conclusion and Future Work

In this chapter we demonstrate how the CCC measure can account for the relative effectiveness of robotic communication methods. We extend the CCC measure to include factors such as the time and energy spent communicating in addition to the resources spent resolving spatial conflicts. We demonstrate the effectiveness of our methods in comparing between very different communication methods falling within categories of no communication, localized and centralized communication methods. By using this information we are able to match the most effective communication scheme to a given robotic domain. We present two general adaptive communication algorithms, uniform and neighborhood methods. We show, in thousands of foraging experiments, that coordination cost is indeed negatively correlated with productivity, and that the use of our adaptive methods leads to significant performance boosts. While we find the neighborhood adaptive method to be more effective in the robotic foraging domain we studied, both approaches are likely to be applicable to many other domains [13, 30, 34, 70]. It is possible that the uniform method is easier to implement or will yield better adaptive qualities in other domains.

For future work, we hope to develop additional applications for our measure on groups of real robots. In the near future, we intend to implement these ideas upon groups of vacuuming robots. In this paper, we studied costs on the robot level. Certain extensions may be necessary when viewing joint team resources, such as shared bandwidth.

Similarly, we are developing expansions to facilitate comparison and adaptation even within heterogeneous groups of robots with diverse qualities. We believe our measure holds promise for further improving group productivity in a variety of domains and group compositions.

Chapter 4

Algorithm Selection for Constraint Optimization Domains

4.1 Introduction

When multiple agents operate within a joint environment, inter-agent constraints typically exist between group members. Assuming these agents operate within a cooperative environment, the team must decide how to coordinate satisfying as many of these constraints as possible [76]. Examples of instances of such problems include classic distributed planning and scheduling domains – problems that are known to be NP-complete problems [47, 52].

Despite the computational complexity inherent in these problems, a variety of algorithms have been suggested for solving these types of problems [12, 47, 48, 52, 60, 76]. These algorithms differ in what and how agents communicate to attempt to find an optimal assignment. Each of these approaches have different resource cost requirements (e.g., time, number of messages), and are often useful in different problem classes. Thus an important task for designers of coordinated multi-agent systems is to find the coordination algorithm that will work best within the coordination problem instance.

In this chapter we claim that an algorithm selection approach is helpful in dictating which type of approach to use. The key to our approach is that differences between algorithms are typically quite large, and can be locally measured. This allows agents to locally control what information to transfer to group members.

To demonstrate the effectiveness of our approach we study two complex coordination domains – a general graph coloring domain [47, 48, 76] and a TAEMS-based scheduling domain [39]. Within each domain we performed thousands of trials involving a variety team sizes and problem parameters. Our results show that our algorithm selection approach was effective in both domains and problem types, significantly outperforming existing methods.

4.2 Using Phase Transitions to aid Algorithm Selection

We claim that algorithm selection within Distributed Constraint Optimization Problems (DCOP) can be based on finding phase transitions within scheduling and planning problem instances. Phase transitions are a well known phenomenon across which problems display dramatic changes in the computational difficulty and solution character [54]. Based on this knowledge, we expect to find attributes that separate between fundamentally different types of problems. Assuming each algorithm is best suited for different clusters of problems, a clear policy will be evident as to which algorithm to select.

4.2.1 Modeling Constraint Satisfaction and Optimization

Following previous DCOP work we define a DCOP problem as a set of variables with each variable being assigned to an agent who has control

of its value. Cooperative agents must then coordinate their choice of values so that a global utility function is optimized. Formally, this process has been described as [47, 53]:

- A set of N agents A = A_1, A_2 ... , A_N
- A set of n variables V = x_1, x_2 ... , x_n
- A set of domains D = D_1, D_2 ... , D_n where the value of x_i is taken from D_i. Each D_i is assumed finite and discrete.
- A set of cost function f = f_1, f_2 ... , f_m where each f_i is a function f_i: $D_{i,1} \times \ldots \times D_{i,j} \rightarrow \mathrm{N} \cup \infty$. Cost functions are also called *constraints*.
- A distribution mapping Q : V $\rightarrow$ A assigning each variable to an agent. Q(x_i) = A_i denotes that A_i is responsible for choosing a value for x_i. A_i is given knowledge of x_i, D_i and all f_i involving x_i.
- An objective function F defined as an aggregation over the set of cost functions. Summation is typically used.

According to this model, the cooperative goal is to minimize the number of constraints in F which are broken. We assume that every agent A_i is solely responsible for dictating the values for the constraints it is responsible for. Thus A_i and x_i are equivalent in this work and can be used interchangeably. Note that the constraint satisfaction domain is a connected problem, as it can be viewed as a special case where the objective function is to find an assignment such that all constraints are satisfied, in place of minimizing the value of F [48].

4.2.2 Phase Transitions within Constraint Satisfaction and Optimization Problems

We claim that large differences between DCOP algorithms are typically apparent, even when agents are confined to using locally available information. We believe that the reason for this lies in different problems belonging to fundamentally different levels of problem difficulty. The basis of this claim is within previous studies who have claimed that NP-complete problems are not all equally difficult to solve [11, 54]. Many instances of NP-complete problems can still be quickly solved, while other similar instances of problems from the same domain cannot.

The concept of phase transitions has been applied to differentiate classes of these "easy" and "hard" NP-complete problem instances [54]. Within distributed constraint satisfaction problems (DCSP), these problems can typically be broken into an easy-hard-easy pattern [54, 48]. The first set of easy problems represent a category of under-constrained problems. All DCSP algorithms typically find an optimal solution quickly for these instances. At the other extreme, the second easy category of problems are those that are over-constrained. Within these problems, the same algorithms can typically demonstrate that no solution exist, and thus these algorithms end in failure. The hardest DCSP problems to solve are those within the phase transition going from under to over-constrained problems, a category of problems also called "critically constrained". These problems are the hardest to solve, with no solution often being found [54].

One may view the DCOP problem as a generalization to the more basic DCSP decision form of the problem. As a result, it would seem that even over-constrained DCOP instances cannot be easily solved

and still comprise "hard" problems [78]. Consequently, DCOP problems should be divided only into Easy-Hard categorizes (instead of Easy-Hard-Easy) or those easy problems to solve before the problem's phase transition, and "hard" problems after this point [59, 78]. However, it has been claimed [59] that certain optimization problems may in fact follow an easy-hard-easy distribution. Even this opinion agrees that problem clusters do exist, and the difference of opinion revolves around the number of these clusters. If we follow the easy-hard model we should expect to see two clusters of problems with a transitionary phase between the two, but following the easy-hard-easy model should yield three such clusters with two transitionary phases.

While DCOP problems are NP-hard for all but the most trivial problems [46], a variety of algorithms can be used for attempting a solution in this domain. These algorithms impact when constraints are communicated between agents, thus impacting how the agents attempt to minimize F. We can formally expand the classic DCOP model into an algorithm selection based model by modeling the selection of algorithms $\{CA_1 \dots CA_j\}$ that each agent can choose in deciding what to communicate while attempting a solution. The intrinsically different approaches used by algorithms $\{CA_1 \dots CA_j\}$ makes them best suited for problems of differing levels of complexity.

This realization significantly simplifies the process of finding those problems instances where a given DCOP algorithm, CA_{a_j}, will be superior to other algorithms within $\{CA_1 \dots CA_j\}$. Instead of viewing all domain problems as an enormous state space where we must map the relative effectiveness of algorithms $\{CA_1 \dots CA_j\}$, we instead focus on finding the problem attributes that differentiate these algorithms. After these attributes have been found, we expect to be able to cluster problems as "easy" or "hard" types of interactions. One type of

algorithm will then be dominant within the easy problems, followed by phase shift(s) where differences between algorithms are smaller and less apparent, followed by another large problem cluster where a second algorithm becomes dominant. As a result, our research focuses on two important questions: 1. What are the attributes that differentiate between algorithms $\{CA_1 \ldots CA_j\}$? 2. At what attribute values should one switch between algorithms?

In order to demonstrate the effectiveness of this approach, we studied two domains: a general graph coloring domain, and the TAEMS scheduling domain. In both domains we found domain attributes that differentiated the algorithms being studied. This allowed us to quickly identify which algorithm could best solve large clusters of problems with only a very short learning period.

4.3 Algorithm Selection in Graph Coloring

Graph coloring problems can be viewed as a generalization of coordination problems. In the 3-color MaxSAT domain, graph nodes (n) are initially randomly assigned one of 3 colors. The goal of this domain is that the nodes coordinate a solution whereby the number of neighbors sharing the same color is minimized (F). Generally, these nodes can be viewed as representing different people or agents, while their various colors on the nodes' edges represent the agent's constraints or preferences that must be coordinated within the group (m).

A range of algorithms exist for solving MaxSAT problems. Wellknow algorithms include distributed breakout (DBO) ([76], asynchronous backtracking (ABT) [75], asynchronous weak-commitment (AWC) [75] and the Optimal Asynchronous Partial Overlay (OptAPO) [47]. According to our thesis, we expected that these algorithms perform

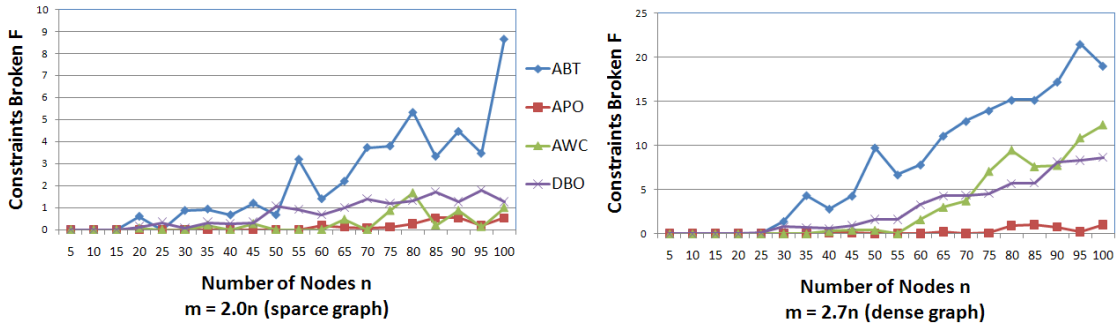

Figure 4.1: Graph coloring performance with ABT, AWC, DBO, and OptAPO algorithms with random graphs with 5-100 nodes (X-axis) and edges = 2.0n (left) and 2.7n (right). Each datapoint represents averaged results from 30 runs with 100 cycles per run.

best in different problem attributes. Within this domain, these attributes include domain parameters such as the number of nodes, edge, time to solve the problem, number of messages and cost of communication.

As DCOP algorithms are inherently distributed, debate exists how performance should be measured. The most common performance measure, which we based our experiments on, is how many cycles were need to solve a given problem instance [75]. Within this measure, one unit of "time" is taken as the series of actions where agents process all incoming messages, process those messages, and send a response. In our experiments, we chose the more accepted cycles based measure to evaluation performance[1].

[1]Other measures besides cycles have been proposed to evaluate the DCOPs' performance. Meisel et al. [36] have argued that performance should be measured based in terms the number of computations each distributed agent performs and proposed a concurrent constraint checks measure (ccc) to quantify this amount. A hybrid measure, proposed by Davin and Modi [12], suggest using a Cycle-Based Runtime (CBR) measure that is parameterized between latency between cycles and computation speed as measured by the concurrent constraint checks measure. Interestingly, the Adopt algorithm [53] outperforms all other algorithms once these measures are used. Note that if one chooses the ccc or CBR measures, or if new DCOP algorithms are found with better performance within the cycle based measure, they can be inserted in place of the algorithms we

We used the Farm simulation environment [28] to create randomized instances of MaxSAT problems. We first varied parameters such as the number of nodes and edges within these problems. Specifically, we studied "sparse" coloring graph problems where the number of edges, m = 2.0n, and "dense" graph problems with m = 2.7n. Traditionally, the sparse problems are thought to belong to the "easy" problem set and the dense problems belonging to the "hard" category [47].

Figure 4.1 represents the performance results of the ABT, AWC, DBO, and OptAPO within this 3-color MAXSat domain. Each algorithm was given 100 time units (cycles) to run, and we measured the number of constraints non-fulfilled after this time (F). We created anytime versions of these algorithms [80] where each algorithm stored the solution minimizing the value of F during task completion. Note that the OptAPO algorithm on average outperformed all other algorithms. This result is consistent with previous finding demonstrating the effectiveness of the OptAPO algorithm regardless of the number of edges or nodes within the problem [47].

However, we found that the best algorithm to use also differed radically based other parameters such as the time allotted to solve a problem instance. In Figure 4.2 we again ran the ABT, AWC, DBO, and OptAPO algorithms but allotted only 10 cycles of runtime. Note how the APO algorithm performs significantly worse than the other algorithms (in problems with > 40 nodes) with the DBO algorithm performing significantly better, especially in dense graphs with more than 60 nodes.

Communication costs can also radically effect which algorithm we should select. For example, OptAPO [47] is a complete algorithm that

studied.

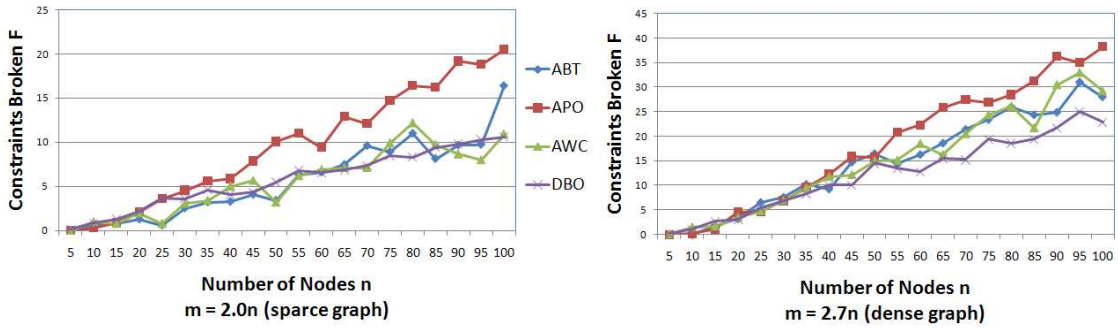

Figure 4.2: Graph coloring performance with ABT, AWC, DBO, and OptAPO algorithms with random graphs with 5-100 nodes (X-axis) and edges = 2.0n (left) and 2.7n (right). Each datapoint represents averaged results from 30 runs with 10 cycles per run.

relays on having nodes communicate their constraints to a mediator node responsible for creating a partially centralized solution. DBO is a hill-climbing algorithm that may never find the global optimal solution for a given graph. However, the localized characteristics of the DBO algorithm allows it to resolve many coloring conflicts without high communication overheads. In fact, the algorithm never communicates beyond its immediate graph neighbors, something other algorithms such as APO rely on. Assuming such communication is costly – say because of security or cost concerns, the OptAPO algorithm should also be avoided even if unlimited time exist to solve these problems.

Figure 4.3 demonstrates the impact of non-local communication cost on algorithm selection. In this graph we compared the performance of the OptAPO and DBO algorithms in dense MaxSAT problems with 100 cycles allotted. When communication was free the APO algorithm (APO-100 Cost 0) did significantly outperform DBO. However, once non-local communication had a cost of 0.02 quality units per communication link, the DBO algorithm outperform OptAPO (APO-100 Cost 0.2)

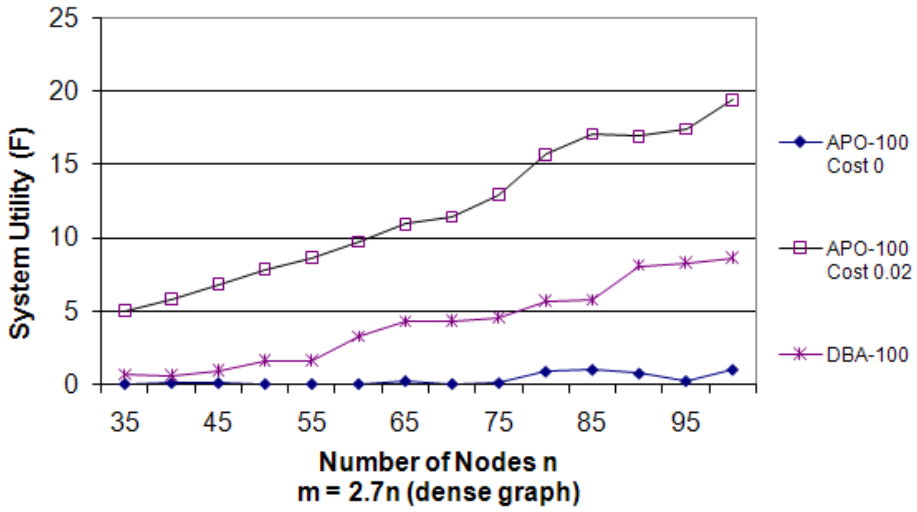

Figure 4.3: The impact of cost on graph coloring algorithms

Because of the radically different performance of these algorithms the selection choice is often quite clear. Let us assume that agents are aware of performance limitations such the time to complete the task, the cost of non-local links, etc. A clear policy typically becomes immediately evident. For example, assume there is no communication cost and agents need to find the best MaxSAT solution given unlimited cycles. OptAPO is clearly the best choice. Conversely, assuming communication is costly, or only a very short period of time is allocated, DBO must be selected.

Figure 4.4 demonstrates the effectiveness of algorithm selection when these problem attributes were known in advance. We generated 100 total MaxSAT problems with random problem attributes (ie. time to solve the problem, number of nodes and edges, and non-local communication costs). Approximately half of these problems could be optimally solved with DBO and OptAPO respectively. When a clear policy was evident, such as when non-local communication was costly or the time to solve problems was very short or very large, our approach definitively selected between DBO and OptAPO. When

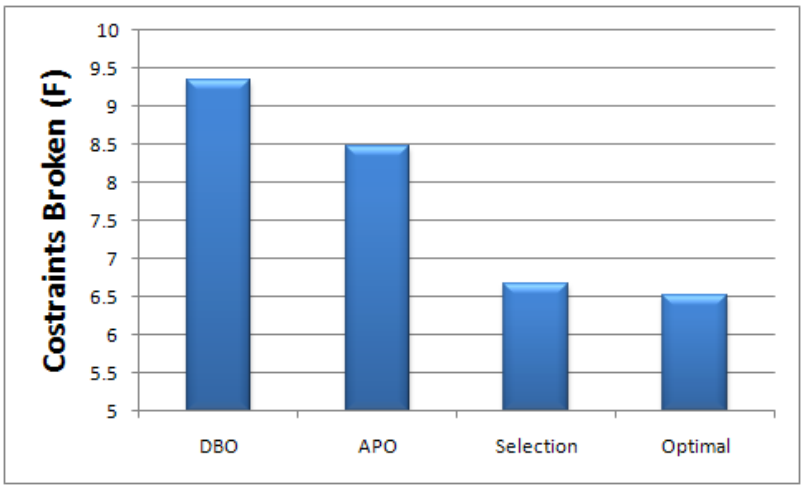

Figure 4.4: Comparing the cost of communication on algorithm selection

no clear policy was defined (as is the case in borderline instances), our approach randomly chose between these algorithms. In order to strengthen the significance of this experiment, we ensured that at least 25 percent of the problem instances were taken from the category when no clear policy existed. Notice that our selection approach closely approximated the optimal choice, and significantly outperformed statically choosing either DBO or OptAPO. We performed a two-tailed t-test comparing our dynamic approach to the static DBO and OptAPO methods. The resulting p-score was well below 0.05, supporting the significance of the presented approach. Similarly, we compared the dynamic approach with the optimal selection policy and found only an insignificant difference (p-score greater than 0.8) between these values. This supports our claim that randomly selecting between algorithms in borderline cases does not significantly hurt performance.

4.4 Algorithm Selection in TAEMS Scheduling Problems

While general DCOP domains provide an excellent formalism and general testbed for different algorithms, it is unclear how many real-world problems can be mapped to these theoretical domains [46]. Even technically, the DCOP model strives to minimize the cost function F, while distributed schedules involve maximizing a group's combined utility. Nonetheless, work by Maheswaran et al. [46] has demonstrated that distributed scheduling problems can be directly mapped to the DCOP framework. Thus, we reasoned that the algorithm selection approach we present should be equally relevant here as well.

However, finding the phase shifts that separate between coordination algorithms within real-world domains is far from trivial [9]. MaxSAT problems have a relatively limited number of problem parameters such as the time to solve a problem, and if communication had a cost. Moreover, constraints typically have equal weighting and every agent has equal numbers of constraints (edges). In contrast, as we now describe, the TAEMS scheduling domain is far more complex.

4.4.1 The TAEMS Scheduling Domain

TAEMS [39] is a framework for formalizing the interrelationships between agents engaged in a group task. This framework has been shown to be effective in quantifying a variety of tasks. Briefly summarized, the TAEMS language is composed of methods, tasks and subtasks that define a coordination problem. Methods represent the most basic action any agent can perform and should have associated with them a list of one or more potential outcomes. This outcome list describes what the quality (Q), duration (D) and cost (C) distributions will

be for each possible result the method might achieve when it is executed. Tasks represent the higher level abstraction and describing the interrelationship between possible actions. This is represented by a quality accumulation function, or QAF, that defines how the quality of the subtasks is used to compute the quality of the task. QAF's are of three forms: min, max or sum. In a min QAF, the total added quality is taken from the minimum from all subtask possibilities – it could be thought of as the logical AND command between tasks. In a max relationship this quality is the maximum value (or the logical OR), and sum is the sum of all subtasks. Finally, hard constraints can be modeled as non-local effects (NLE's) that *enable* or *disable* other tasks. Soft constraints are modeled through *facilitates* or *hinders* relationships. For example, assuming one task must occur before another, we could state that the task enables the other. Assuming both tasks cannot be performed we could say one task disables the other.

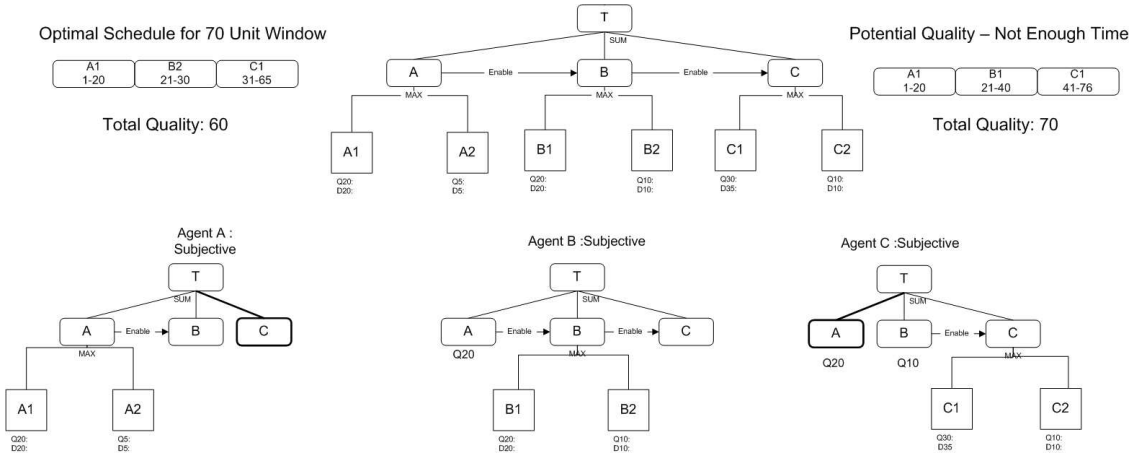

Figure 4.5: A sample TAEMS scheduling problem. Note that the optimal solution is not evident by greedy selection.

Figure 4.5 is an example of a scheduling problem instance, described in TAEMS statements. In this example, three agents, A, B, and C must coordinate their actions to find the optimal schedule for task

T. Task T has three subtasks (A, B, and C) and these tasks joined by a sum relationship. There are enable relationships between these tasks and thus they must be executed sequentially. In this example, an optimal schedule would be for A to schedule method A1, B to schedule B2, and C to schedule C1. There is insufficient time for B to schedule method B1 and for A to schedule A1 so one of these agents must sacrifice scheduling the method with the highest quality so the group's quality will be maximized. However, each agent only has a local view as the methods and tasks with which it is directly involved, and thus is unaware of the others' constraints.

TAEMS scheduling problems can be generated with many more constraint types. These include factors such as: the number of tasks to be scheduled, the hierarchical structure of these tasks, the number of agents able to perform each task, the number of NLE's between tasks, overlapping effects of task windows due to their duration, and redundancy effects if tasks have multiple possible ways to be performed.

4.4.2 Algorithms for Coordinating TAEMS Decisions

Nonetheless, we claim the algorithm selection process can be addressed through an algorithm selection process. As was the case in the DCOP formalism, each agents with TAEMS only has a partial view of its constraints within the distributed system. Different algorithms $\{CA_1 \dots CA_j\}$ control how much information is sent, and thus if part(s) of the problem are solved centrally or locally. Thus, each agent can be thought of as a value that controls how much information to send in solving the problem. Assuming each node uses algorithms that locally attempt a solution very little information will be send. Conversely if every value indiscriminately sends all constraints a centralized solution will

be attempted.

The utility of each agent's choice is evaluated as follows. First, each agent chooses from algorithms $\{CA_1 \ldots CA_j\}$ to decide what information to send. This information is then sent to a Constraint Optimization Problem (COP) solver. We refer the reader to [22] for implementation details of the constraint programming approach used by the this COP solver. To calculate the group's schedule, each agent first sent its TAEMS statements to the COP solver. The COP solver then processed the total constraint information sent by all agents and returned the utility of the resulting schedule. As we assume communication has some cost, the group's total utility is the gain from the COP's generated schedule minus the sum of all communication costs of all agents.

We considered the following library of algorithms. CA_1, or the Exhaustive algorithm, requires an agent to communicate all of their constraints. As this algorithm sends all information it is guaranteed to send enough information to find an optimal coordination solution, and thus can have the highest gain. However, this algorithm also has the highest communication cost. Furthermore, this algorithm may not yield any quality before finding an optimal solution. Assuming the problem is over-constrained, even effective COP solvers cannot find easily find any solution, and may yield zero quality at the end of the search window.

At the other extreme, algorithm CA_2 or the Approximation algorithm, makes local decisions and informs the COP solver as to its choice. This algorithm was inspired by Durfee's [15] comment that "ignorance is bliss", i.e., having agents send all information at their disposal may not be advantageous. Referring back to the example in Figure 4.5, agents can greedily select the option with the highest

quality. In this example, agents would choose their first options, A1, B1, and C1. The COP scheduler would then be forced to forgo the last option C1, and return a quality of 40 (well below the optimal solution of 60). This algorithm guarantees a relatively low cost as it communicates only the minimum amount. Especially if the problem is over-constrained, this approach may perform better than the Exhaustive approach as it can typically simplify the problem to the level that the COP scheduler will return some quality, even if it may be non-optimal.

Next, we also implemented an existing algorithm selection processes based on the work of Allen and Minton [2] (CA_3). Previously, they suggested an approach where the best algorithm is selected through running all algorithms in the library for a short period of time, and then selecting the best algorithm based on secondary performance characteristics compiled from this preliminary trial. Following this approach, this algorithm first runs algorithm CA_1 for a short period of time. If the CA_1 does not return any quality within that test period, the problem is deemed over-constrained and CA_2 is used in the remaining time. Otherwise, CA_1 is allowed to continue. Assuming communication is not free, this approach may have the composite costs of both CA_1 and CA_2, and thus may not perform well. However, assuming communication is free, it may produce the optimal selection by trying all algorithms in the library.

4.5 Scheduling Tightness Model

In order to simplify finding phase shifts, we present a tightness measure to quantify the complexity of agent interactions in this scheduling domain. We formalize an agent's TAEMS scheduling problem as fol-

lows:

Similar to the DCOP formalization let $G = \{A_1, A_2 \ldots, a_N\}$ be the group of N agents trying to maximize their group's collective scheduling utility. Each agent has a set of m tasks, T = $\{T_1, \ldots, T_m\}$ that can be performed by that agent. Each Task, T_i, has a time **Window** where the task needs to be performed, a **Duration** as the length of time the task requires to be completed, and a **Quality** as the utility that task will add to the group upon its successful completion. While task window lengths are typically fixed within this domain, task duration and quality can be variable. Let us model W_i as the fixed Window length for task i, $\{R_{i1}, R_{i2}, \ldots, R_{ij}\}$ as the possible duration lengths for a given task T_i and $\{Q_{i1}, Q_{i2}, \ldots, Q_{ij}\}$ as the corresponding quality amounts for these options.

We denote a task's tightness as:

Tightness$(T_i) = \frac{Duration_{max}(T_i)}{Window(T_i)}$

where $Duration_{max}(T_i)$ returns the maximal duration from $\{R_{i1}, R_{i2}, \ldots, R_{ij}\}$.

This tightness measure has two key qualities: First, it is locally measurable. Any agent can measure its task tightness without any additional input from other task agents. This allows agents to effectively autonomously select the best coordination algorithm. Second, it can effectively quantify the impact making a local decision will have. By definition, a tightness of 1.0 or less means that the problem is not constrained, as no task option exceeds the task window naturally allotted for that task. As a result, that agent can safely make a local decision (CA_2) without fear that the optimal solutions will not be found. However, once this measure exceeds 1.0, the risk exists that tasks overlap, and the optimal solution involves one task being sacrificed in favor of another.

4.6 Evaluating Algorithm Selection Scheduling Policy

The tightness measure presented in the previous section was crucial in identifying when to select which scheduling algorithm. In this section we describe how three classic problem clusters were quickly found based on this measure: under-constrained, constrained, and over-constrained. Each cluster corresponded to a clear coordination algorithm choice – creating an effective algorithm selection policy. We then evaluated the effectiveness of this policy in randomly generated scheduling problems with a variety of communication costs. We found the trained approach significantly improved performance in cases where communication had a cost, always coming within an insignificant amount of the optimal selection.

4.6.1 Training the Scheduling Algorithm Selection Policy

In order to train the algorithm selection policy, we randomly generated 50 problems for training which algorithm should be selected. We used a problem generator created by Global Infotech Inc. (GITI) for the purpose of generating TAEMs problems within the framework of the the COORDINATORS DARPA/IPTO program[2]. We created problems where TAEMS parameters, such as the number of tasks to be scheduled, the hierarchical structure of these tasks, the number of agents able to perform each task, number of NLE relationships between tasks, overlapping effects of task windows due to their duration, and redundancy effects as described above, were randomly selected. Note that these 50 problems represent a small fraction of the total number of the thousands of problem permutation the program's sce-

[2]http://www.darpa.mil/ipto/programs/coordinators/

nario generator could create. After these problems were created, every agent computed its average task tightness based on its local information.

We then ran algorithms CA_1 and CA_2 on this training set. We clustered problems by their relative tightness, ie. problems with tightness 0.0 - 0.5, 0.5 - 1.0, 1.0 - 1.5, etc. Because the purpose of this experiment was to attempt to learn when to select each of these algorithms, we did not also run the algorithm selection process CA_3 at this stage. In this experiment, we also assumed that communication had no cost and allowed the COP scheduler to run for 10 seconds after each local agent decided what information to send. The results of this experiment can be found in Figure 4.6.

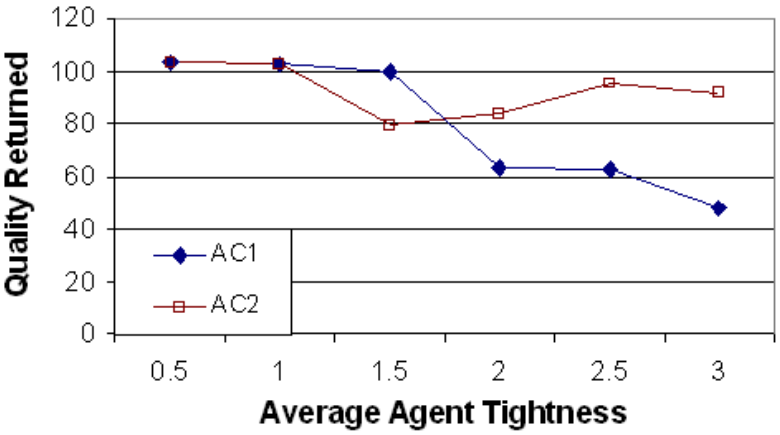

Figure 4.6: Comparing scheduling utility yielded from algorithms that involve agents forwarding all information (CA_1) and local decisions (CA_2) in problems of different tightness complexity

As we see from Figure 4.6, three distinct problem types emerge, under-constrained, constrained, and over-constrained problems. For problems of tightness 1.0 or less, algorithms CA_1 and CA_2 perform equally well. One could term these problems under-constrained, as we found that sending no constraint information (CA_2) is equally effective

to sending all information. Thus, assuming communication has some cost, CA_2 is clearly the better choice in these problems. In problems with a tightness between 1.0 and 1.5, the complete algorithm, CA_1, performed best. These problems were constrained, yet able to be solved even if all constraints were sent to the COP solver. Thus information was useful in finding a higher scheduling utility. After this point (at approximately tightness = 1.75) problems become over-constrained, and added information does not add value. Once again algorithm CA_2 should be selected. While the first phase shift (before tightness 1.0 and after) is based on the definition of our tightness measure, we recognize that the second phase shift (around tightness 1.75) is likely based on the specific COP solver being used. It would not be surprising if other constraint-based solvers would be released in the future that could effectively solve problems even when their average tightness is significantly more than 1.75. However, as these problems are NP-complete we believe some point will always exist separating these types of problems.

Once we quickly identified the approximate tightness value to use each algorithm, an algorithm selection policy could be formed. When agents measured a tightness of 1.0 or less CA_1 was used. CA_2 was selected when the tightness was less than 1.5, and CA_1 was again used when the tightness was greater than 2.0. In the transition between CA_1 and CA_2 we evaluated two possibilities: random choice or midpoint. Within the random choice option, CA_1 or CA_2 was randomly chosen between 1.5 and 2.0. Within the midpoint choice the range between tightness 1.5 and 2.0 was split with CA_1 used until 1.75 and CA_2 after this point.

4.6.2 Evaluating Scheduling Policy with and without Cost

We then created 100 new TAEMS problems with randomly generated parameters, and compared the quality obtained from using algorithms ($\{CA_1 \ldots CA_3\}$) to the algorithm selection policy. We again allotted the COP solver 10 seconds to run each algorithm. Algorithms CA_1, CA_2 and the selection algorithms (midpoint and random) selected what information to send immediately and let the COP solver run for the full 10 seconds. Recall that CA_3 is itself an algorithm selection process where CA_1 is first allowed to run for a short time before deciding to continue with this algorithm or to switch to CA_2. Thus, CA_3 first attempted to find an optimal solution with CA_1 within the first 2 seconds. If no solution was found within this time, it used CA_2 with the remaining time.

First, we consider the case where communication is free. The results of this experiment are in Figure 4.7. As expected, algorithm CA_3 did produce the highest quality, nearly making the optimal selection in all cases (58.6 average quality for CA_3 versus 58.76 in the optimal choice). Both the random and midpoint algorithm selection policies perform similarly (the p-score from within a two tailed t-test comparing these two selection approaches was 0.65) and thus we print the midpoint approach which performed slightly better. Additionally, while CA_3 did outperform our algorithm selection approach in this case, the difference was not significant (p-scored within a two tailed t-test comparing midpoint and CA_3 being 0.67).

However, in many real-world cases, one can assume that communication will have some cost. In these cases, it is critical that agents send as little information as possible, and approaches such as CA_3 that send multiple rounds of information are likely to perform worse than

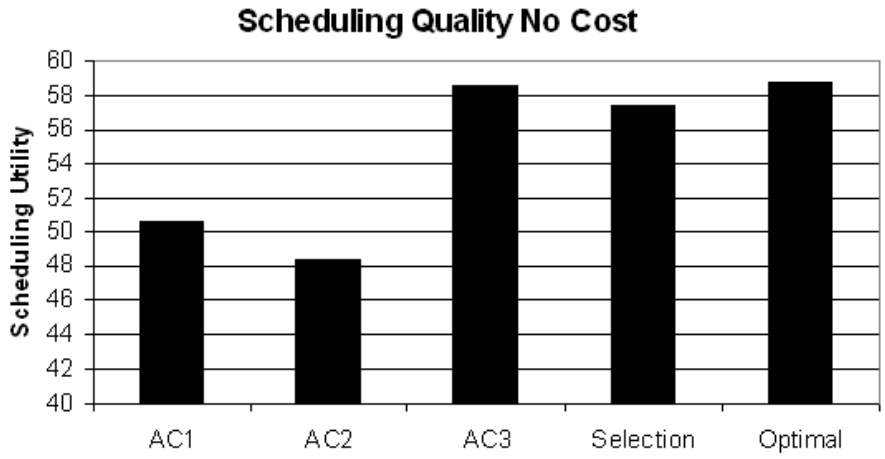

Figure 4.7: Comparing average scheduling utility yielded from algorithms with no communication cost

other approaches. However, the correct policy is likely to depend on this cost. In an extreme example, assuming communication costs are made sufficiently high CA_2 will always dominate. Assuming communication has a more moderate cost, the question becomes what is the crossover between between the coordination algorithms in our library.

Fortunately, the policy for cases with communication can be easily adjusted in this domain. Due to the structure of the TAEMS problems being studied, agents could assume their contribution to the group's utility was roughly inversely proportional to the number of task nodes (1/total-nodes). This being the case, once any local agent knows how many other nodes are within the problem, it could estimate the value of communicating information. For example, say a local node knows communication costs 0.05 per message, and there are 20 total nodes in the problem. Let us assume that agent measures a local tightness of 1.5. Using the cost-free policy, it can estimate that the difference in utility gain of using CA_1 versus CA_2 is approximately 20 for this point (see Figure 4.6). Thus, it estimates that the gain from its information is approximately 1 unit of gain (the difference of 20 units

split over all 20 nodes), while its cost of communication is only 0.05 per TAEMS constraint. Assuming it has under 20 constraints to send, it will choose CA_1. However, assuming the same cost of communication, but a local tightness of 1.75 is measured, the agent estimates there will be no utility gain through using CA_1, but it incur a higher communication cost through using this algorithm. Thus, it will opt for CA_2. We concede that this simplified approach may not apply to more heterogenous problems, and we hope to study such problems at greater length in the future.

As expected, our algorithm selection approach was much more effective than all other approaches once these costs were considered. Figure 4.8 displays the result from the same 100 evaluation cases from the previous experiment, but assumes a cost of 0.05 must be deducted for each TAEMS statement sent. Note how the algorithm selection process significantly outperforms all other option (p-scores being 0.0001 or under) and again comes within an insignificant amount of the optimal selection (p-score from a two tailed t-test again being 0.64). Thus, we found this approach to be extremely effective in cases where communication had a cost.

4.7 Conclusion and Future Work

In this work we present an algorithm selection approach for solving DCOP problems. We focused on two factors: what attributes differentiate between coordination algorithms, and how can we build a selection policy based on those attributes. We present strong empirical evidence of the success of this approach in scheduling and graph coloring domains, suggesting the generality of this work.

For future work, a several directions are possible. In this work,

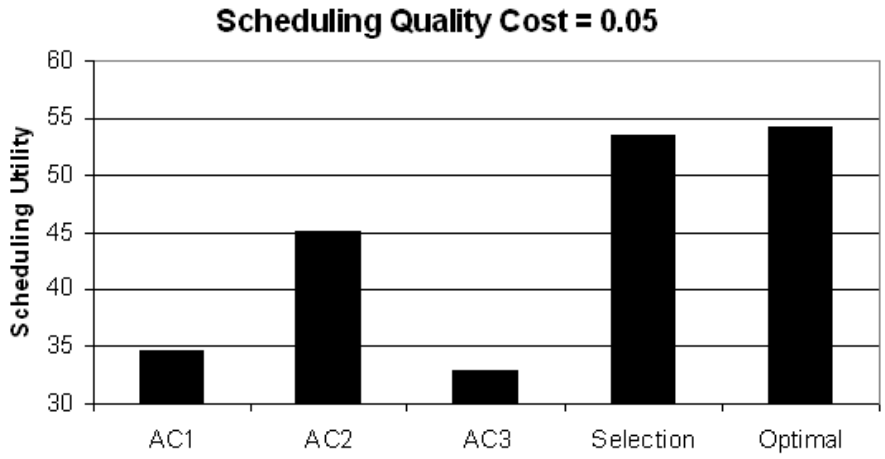

Figure 4.8: Comparing average scheduling utility yielded from algorithms with communication cost of 0.05 units per TAEMS message

we manually found the attributes that differentiated the coordination algorithms within the domains we studied. We hope to study how algorithms can be created to automate this process so that other novel interaction measures, such as the tightness measure we present, may be learned for quantifying coordination interactions in other domains.

The success of our coordination selection approach was rooted in the realization that different clusters of problems can be created based on the hardness of different agent interactions. We drew upon the "phase transition" concept used to describe some constraint satisfaction problems [54]. However, following Brueckner and Parunak [9] we reserve the term "phase transition" to refer to a term used by physicists for mathematically describable behavior within the system, and instead term the clusters of problems we empirically observed as phase shifts. We hope to study in the future what formal models of coordination are possible that can predict where and when these transitions should occur. We believe this study could strengthen the theoretical basis of the work we present.

Chapter 5

Adaptive Full-text Search in Peer to Peer Networks

5.1 Introduction

Full-text searching, or the ability to locate documents based on terms found within documents, is arguably one of the most essential tasks in any distributed network [41]. Search engines such as Google [1] have demonstrated the effectiveness of centralized search. However, classic solutions also demonstrate the challenge of large-scale search. For example, a search on Google for the word, "a", currently returns over 10 billion pages [1].

In this chapter, we address the challenge of implementing full-text searches within peer-to-peer (P2P) networks. Our motivation is to demonstrate the feasibility of implementing a P2P network comprised of resource limited machines, such as handheld devices. Thus, any solution must be keenly aware of the following constraints: **Cost** - Many networks, such as cellular networks, have cost associated with each message. One key goal of the system is to keep communication costs low. **Hardware limitations** - we assume each device is limited in the amount of storage it has. Any proposed solution must take this

limitation into consideration. **Distributed** - any proposed solution must be distributed equitably. As we assume a network of agents with similar hardware composition, no one agent can be required to have storage or communication requirements grossly beyond that of other machines. **Resilient** - our assumption is that peers are able to connect and disconnect at will from the network. As a result, our system must be able to deal with peer failures, a concept typically

To date, three basic approaches have been proposed for full-text searches within P2P networks [74]. Structured approaches are based on classic Information Retrieval theory [24], and use inverted lists to quickly find query terms. However, they rely on expensive publishing and query lookup stages. A second approach creates super-peers, or nodes that are able to locally interact with a large subset of agents. While this approach does significantly reduce publishing costs, it violates the distributed requirement in our system. Finally, unstructured approaches involve no publishing, but are not successful in locating hard to find items [74]

In this paper we present PHIRST, a system for **P**eer-to-Peer **H**ybri**d** **R**estricted **S**earch for **T**ext. PHIRST is a hybrid approach that leverages the advantages of structured and unstructured search algorithms. Similar to structured approaches, agents publish terms within their documents as they join or add documents to the P2P network. This information is necessary to successfully locate hard-to-find items. Unstructured search is used to effectively find common terms without expensive lookups of inverted lists. Another key feature in PHIRST is its ability to restrict the number of peer addresses stored within inverted lists. Not only does this insure that the hardware limitations of agent nodes are not exceeded, it also better distributes the system's storage. We also present a full-text query algorithm where nodes ex-

plicitly reason based on estimated search costs about which search approach to use, reducing query costs as well. Finally, we present how storing redundant copies of these entries can effectively deal with temporary node failures without use of any centralized mechanism.

To validate the effectiveness of PHIRST, we used a real web corpus [58]. We found that the hybrid approach we present used significantly less storage to store all inverted lists than previous approaches where all terms were published [41, 74]. Next, we used artificial and real queries to evaluate the system. The artificial queries demonstrated the strengths and limitations of our system. The unstructured component of PHIRST was extremely successful in finding frequent terms, and the structured component was equally successful in finding any term pairs where at least one term was not frequent. In both of these cases, the recall of our system was always 100%. The system's performance did have less than 100% recall when terms of 2 or more words of medium frequency were constructed. We present several compensatory strategies for addressing this limitation in the system. Finally, to evaluate the practical impact of this potential drawback, we studied real queries taken from IMDB's movie database (www.imdb.com) and found PHIRST was in fact effective in answering these queries.

5.2 Related Work

Classical Information Retrieval (IR) systems use a centralized server to store inverted lists of every document within the system [24]. These lists are "inverted" in that the server stores lists of the location for each term, and not the term itself. Inverted lists can store other information, such as the term's location in the document, the number of occurrences for that term, etc. Search results are then returned by

intersecting the inverted lists for all terms in the query. These results are then typically ranked using heuristics such as TF/IDF [32]. For example, if searching for the terms, "family movie", one would first lookup the inverted list of "family", intersect that file with that of "movie", and then order the results before sending them back to the user.

The goal of a P2P system is to provide results of equal quality without needing a centralized server with the inverted lists. Potentially, the distributed solution may have advantages such as no single point of failure, lower maintenance costs, and more up-to-date data. Toward this goal a variety of distributed mechanisms have been proposed.

Structures such as Distributed Hash Tables (DHTs) are one way to distribute the process of storing inverted lists. Many DHT frameworks have been presented, such as Bamboo [62], Chord [55], and Tapesty [79]. A DHT could then be used for IR in two stages: publishing and query lookups. As agents join the network, they need to update the system's inverted lists with their terms. This is done through every agent sending a "publish" message to the DHT with the unique terms it contains. In DHT systems, these messages are routed to the peer with the inverted list in LogN hops, with N being the total number of agents in the network [55, 62]. During query lookups, an agent must first identify which peer(s) store the inverted lists for the desired term(s). Again, this lookup can be done in LogN hops [55, 62]. Then, the agent must retrieve these lists and intersect them to find which peer(s) contain all of the terms.

Li et al. [41] present formidable challenges in implementing both the publishing and lookup phases of this approach in large distributed networks. Assuming a word exists in all documents, its inverted list will contain N entries. Thus, the storage requirements for these

inverted lists are likely to exceed the hardware abilities of agents in these systems. Furthermore, sending large lists will incur a large communication cost, even potentially exceeding the bandwidth limitation of the network. Because of these difficulties, they concluded that naive implementations of P2P full-text search are simply not feasible.

Several recent developments have been suggested to make a full text distributed system viable. One suggestion is to process the structured search starting with the node storing the term with the fewest peer entries in its inverted list. That node then forwards its list to the node with the next longest list, where the terms are locally intersected before being forwarded. This approach can offer significant cost savings by insuring that no agent can send an inverted list longer than the one stored by the **least** common term [74]. Reynolds and Vahdat also suggest encoding inverted lists as Bloom filters to reduce their size [61]. These filters can also be cached to reduce the frequency these files must be sent. Finally, they suggest using incremental results, where only a partial set of results are returned allowing search operations to halt after finding a fixed number of results, making search costs proportional to the number of documents returned.

Unstructured search protocols provide an alternative that is used within Gnutella and other P2P networks [10]. These protocols have no publishing requirements. To find a document, the searching query sends its query around the network, until a predefined number of results have been found, or a predefined TTL (Time To Live) has been reached. Assuming the search terms are in fact popular, this approach will be successful after searching a fraction of the network. Various optimizations have again been suggested within this approach. It has been found that random walks are more effective than simply flooding the network with the query [44]. Furthermore, one can initiate mul-

tiple simultaneous "walks" to find items more quickly, or use state-keeping to prevent "walkers" from revisiting the same nodes [44]. Despite these optimizations, unstructured searches have been found to be unsuccessful in finding rare terms [10].

In super-peer networks, certain agents store an inverted list for all peer documents for which it assumes responsibility. Instead of publishing copies over a distributed DHT network, agents send copies of their lists to their assigned super-peers. As agents are assumed to have direct communication with its super-peers, only one hop is needed to publish a message, instead of the LogN paths within DHT systems. During query processing, an agent forwards its request to its super-peer, who then takes the intersection between the inverted lists of all super-peers. However, this approach requires that certain nodes have higher bandwidth and storage capabilities [74] – something we could not assume within our system.

Hybrid architectures involve using elements from multiple approaches. Loo et al. [42, 43] propose a hybrid approach where a DHT is used within super-peers to locate infrequent files, and unstructured query flooding is used to find common files. This approach is most similar to ours in that we also use a DHT to find infrequent terms and unstructured search for frequent terms. However, several key differences exist. First, their approach was a hybrid approach between Gnutella ultrapeers (super-peers) and unstructured flooding. We present a hybrid approach that can generically use any form of structured or unstructured approaches, such as random walks instead of unstructured flooding or global DHT's instead of a super-peer system. Second, in determining if a file was common or not, they needed to rely on locally available information from super-peers, and used a variety of heuristics to attempt to extrapolate this information for the global network [42].

As we build PHIRST based on a global DHT, we are able to identify rare-items based on complete information. Possibly most significantly, Loo et al. [43] only published the files' names, and not their content. As they considered full text search to be infeasible for the reasons previously presented [41], their system was limited to performing searches based on the data's file name, and not the text within that data. As our next section details, we present a publishing algorithm that actually becomes cheaper to use as subsequent nodes are added. Thus, PHIRST is the first system to facilitate effective full-text search even within large P2P networks.

5.3 PHIRST Overview

First, we present an overview of the PHIRST system and how its publishing and query algorithms interconnect. While this section describes how information is published within the Chord DHT [55], PHIRST's publishing algorithm is generally presented in section 5.4 so it may be used within other DHT's as well. Similarly, section 5.5 presents a query algorithm (algorithm 4) which generally selects the best search algorithm based on the estimated cost of performing the search algorithms at the user's disposal. The selection algorithm is generically written such that new search algorithms can be introduced without affecting the algorithm's structure. Only later, in algorithm 5 do we present how these costs are calculated specific to the DHT and unstructured search algorithms we used.

In order to facilitate structured full-text search for even infrequent words, search keys must be stored within structured network overlays such as Chord. Briefly, Chord uses consistent hash functions to create an m-bit identifier. These identifiers form a circle modulo 2^m. The

node responsible for storing any given key is found by using a preselected hash function, such as SHA-1, to compute the hash value of that key. Chord then routes the key to the agent whose Chord identifier is equal to or is the successor (the next existent node) of that value [55]. For example, Figure 5.1 is a simple example with an identifier space of 8, and 3 nodes. Assuming the key hashes to a value of 6, that key needs to be stored on the next node within the circular space, or node 0. Assuming the key hashes to 1, it is stored on node 1.

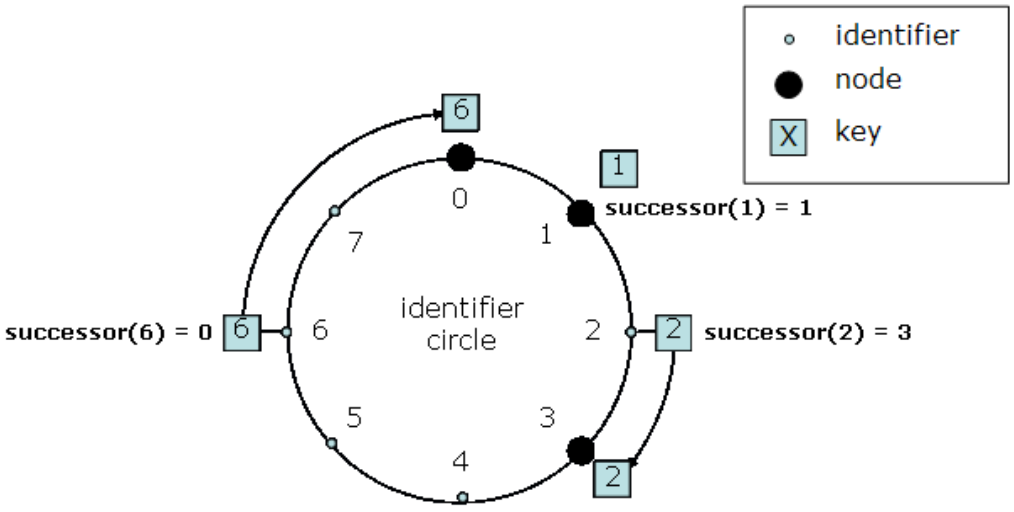

Figure 5.1: An example of a Chord ring with m=3

The hashing quality within the Chord algorithm has several important qualities. First, it creates important performance guarantees, such as LogN average search length. Furthermore, nodes can be easily added (joins) or removed (disjoins) by inserting them into the circular space, and re-indexing only a fraction of the pointers within the system. Finally, the persistent hashing function used by Chord has the quality that no agent will get more than O(LogN) keys than the average [55]. We refer the reader to the Chord paper for further details [55].

However, the DHT's performance guarantees only balancing the

Table 5.1: Example of several words (keys within the DHT), and their inverted lists.

Word (key)	Address1	Address2	Address3	Address4	Address5	Address6	Address7
a	111-1111	111-1112	111-1113	111-1114	111-1115	111-1116	111-1117
aardvark	111-4323						
the	111-1111	111-1112	111-1113	111-1114	111-1115	111-1116	111-1117
zoo	123-4214	123-9714	333-9714				
zygote	548-4342						

number of keys stored per node, but not the number of addresses stored in the inverted lists for each key. For example, Table 5.1, gives an example of the inverted lists for five words. Common words, such as "a" and "the" within the table, will produce much long inverted lists, than uncommon words such as "aardvark" and "zygote". Due to space restrictions we will only present up to the first 7 inverted entries for each word, out of a potential length of N rows. Balancing guarantees only apply to the number of words (out of N), but not the size of each inverted list (the length of that row). Because word distribution within documents typically follow Zipf's law, some of the words within documents occur very frequently while many others occur rarely [33]. In an extreme example, one node may be responsible for storing extremely common words such as "the" and "a", while other nodes are assigned only rare terms. Thus, one key contribution of this paper is a publishing algorithm that can equitable distribute these entries by allowing agents to cap the number of inverted list entries they will store.

Once the publishing stage has been begun, a distributed database exists to search the network for full-text queries. We define the search task as finding a number of results, T, that match all query terms within the documents' text. Capping a query at T results is needed within unstructured searches, as there is no global mechanism for

knowing the total number of matches [74]. Finding only a limited number of results has also been previously suggested within structured searches to reduce communication costs [61]. The second key contribution of this paper is a novel querying algorithm that leverages between structured and unstructured searches to effectively find matches despite the limit in the amount of data each peer stores.

5.4 The Publishing Algorithm

Every time an agent joins the network, or an existing agent wishes to add a new document, it must publish the words in its document(s) as described in Algorithm 5.4. First, the agent generates a set of *max* terms it wishes to add (line 1). Similar to other studies [74] we assume that the agent preprocesses its document to remove extraneous information such as HTML tags and duplicate instances of terms. Stemming, or reducing each word to its root form, is also done as it has been observed to improve the accuracy of the search [74]. Furthermore, as we detail in the Experimental Results section (section 5.7), stemming also further reduces the amount of information needed to be published and stored. The publishing agent, ID_{Source}, then sends every unique term, $Term_i$, to be stored in an inverted list on peer ID_{DEST} (lines 3-4). The keys being stored are these words that are sent, with each word either creating a new inverted list, or being added to an existing file. In addition to these terms, the agent also updates a counter of the total number of documents contained between all agents within the system (line 4). For simplicity, let us assume this global counter is stored on the first agent, ID_1. We will see that this value is needed by the query algorithm described below.

PHIRST's publishing algorithm enforces an equitable term distrib-

Algorithm 3 Publishing Algorithm(Document Doc)

1: Terms $\Leftarrow$ Preprocessed words in Doc
2: **for** $i = Term_1$ to $Term_{max}$ **do**
3: PUBLISH($Term_i$, ID_{Source}, ID_{DEST})
4: PUBLISH(DOC-COUNTER+1, ID_1)
5: **for** $i = Term_1$ to $Term_{received}$ **do**
6: **if** SIZE(ID_{DEST}, $Term_i$) $<$ d **then**
7: ADD-Term($Term_i$, ID_{Source})
8: UPDATE-Counter($Term_i$, $COUNTER$)

ution by only storing inverted lists until a length of d. For every term node, $Term_i$ out of a total of $received$ terms, ID_{DEST} is requested to store it must decide if it should fulfill that request. As lines 6 and 7 of the algorithm detail, assuming agent ID_{DEST} currently has fewer than d entries for $Term_i$, it adds the value ID_{Source} to its list (or creates an inverted list if this is the first occurrence). Either way, nodes log that a certain number of $COUNTER$ instances of that term exist (lines 8). This information is used by the query algorithm to determine the global frequency of this term. Because we limit each node to only storing d out of a possible N terms, the storage requirements of the system are reduced to d*N from N*N. As we set $d <<$ N, we found this savings to be quite significant.

Theoretically, additional information about each term may be published, such as the position that term occurred or how many instances of that term existed within the document and aggregate this and similar information into a rating for the term it is about to publish. This information may be especially important when more than d instances of that term exist. The receiving agent, ID_{DEST}, could then decide which d term instances to store by continuously sorting scores of the terms it has, and maintaining only those with the top d highest rating. In a similar vein, if more than d instances of $Term_i$ exist, it may

be advantageous to store the d most recent documents, especially if turnover exists within nodes.

The performance guarantees of DHT's such as Chord insure the publishing algorithm runs with fairly low cost. Because each node, ID_{Source}, needs LogN hops to find the agent, ID_{DEST}, responsible for storing that term's inverted list, the total number of messages needed to publish a document is of order $O(max * logN)$ where max is the number of terms in that document. Note that the publishing algorithm described here sends all terms, even those which in fact do not need publishing because they already contained d terms.

5.5 The Search Algorithm

The search algorithm is called once any agent wishes to conduct a distributed full-text search. As Algorithm 4 describes, this process operates in two stages. First, we retrieve the global frequencies of all search terms (line 1) and sort all terms from least to most frequent (line 2). This value can be calculated through looking up the frequency of that term ($COUNTER$), and dividing this number by the total number of documents ($DOC - COUNTER$). Finding these values requires one lookup of the value of $DOC - COUNTER$ (assumed to be stored on agent ID_1 in the publishing algorithm), as well as a lookup for the frequencies of each term from the agent storing term $Term_i$. Referring back to algorithm 5.4 note that the peer storing $Term_i$ has a counter with this value even if more than d instances of this term occurred.

Once the frequency of all terms are known, the algorithm then reasons about which algorithm to select. This process iteratively calls the tradeoff function which we define below (algorithm 5). If unstruc-

Algorithm 4 Hybrid Search Algorithm(String $Query_1 \ldots Query_{max}$)

```
1: space ⇐ ∞ {Used for initialization to all P2P nodes}
2: Retrieve Frequencies of Query_1 ... Query_max
3: Term ⇐ Sorted Query Terms Least to most Frequent {Term is an array}
4: for i = Term_1 to Term_max do
5:    Frequency ⇐ Product of Frequencies(Term_i ... Term_max)
6:    Tradeoff ⇐ Calculate-Tradeoff(space, Term_i ... Term_max, Frequency)
7:    if Tradeoff > 0 then
8:       while Found < T AND NOT Exhausted(space) do
9:          Search-Unstruct(space, Term_i ... Term_max)
10:      Break
11:   else
12:      space ⇐ List(Term_i) ∩ space
13:      if i=Term_max then
14:         if space > T then
15:            return first T list entries
16:         else
17:            return all list entries
```

tured search is deemed less costly, all terms are immediately searched for simultaneously (lines 7–10). This type of search can either terminate because T matches have been found or the search space has been exhaustively searched. If structured search is deemed less costly, that term's inverted list is requested, and the search space is intersected with that of the new term (line 12). Assuming we have reached the last term (lines 12-17) we return the first T matches found after all terms were successfully intersected. Once the structured search identifies that fewer matches than T matches were found (line 15) it returns all list entries (line 17).

This algorithm has several key features. First, the search process is begun starting with the least frequent term. This is done following previous approaches [74] to save on communication costs. We denote the inverted list length of the **least** common search term as length($Term_1$)

where length is a function that returns the size of an inverted list and $Term_1$ is the first term after the terms are sorted based on frequency. Each successive peer receives the previously intersected list, and locally intersects this information with that of its term (line 13). The result of this process is that intersected lists become progressively smaller (or at worse case stay the same size) with the maximum information any peer can send being bounded by length($Term_1$). Second, one might question why agents do not immediately return the entire inverted list of the terms they store, instead of first returning the term's frequency. This is done because the information gained from this frequency information, such as bounding search costs to the size of the least frequent term, far outweighs the search costs involved with processing the query in two stages. Finally, as the search goal is to return T results, the last node within a structured search does not need to return its entire inverted list. Instead, it only needs to send the first T results (or failure or NULL as in line 17 if under T results exist). Because of this, the maximal search cost will be of order (max-1) * length($Term_1$) + T where max is the number of terms in the search query.

Arguably the most important feature of this algorithm is its ability to switch between using structured and unstructured searches midway through processing the query terms. Even if structured search is used for the first term(s), the algorithm iteratively calls the tradeoff algorithm (algorithm 5) after each term. Once the algorithm notes that unstructured search is cheaper, it immediately uses this approach to find all remaining terms. For example, assume a multi-word query contains several common and uncommon words. The algorithm may first take the intersection of the inverted lists for all infrequent words to create a list f. The algorithm may then switch to use unstructured search within f to find the remaining common words.

Similarly, note that this approach lacks a TTL (Time To Live) for its unstructured search. We assume unstructured searches are to be used only when the expected cost of using an unstructured search is low (see algorithm 5 below). We expect this to occur when the unstructured search will terminate quickly, such as when: (i) the search terms are very common from the onset or (ii) unstructured search is used to find the remaining common terms after structured search generated an inverted list of f terms.

We now turn to the search specific mechanism needed to identify which search types will have the higher expected cost. This tradeoff depends on T, or the number of search terms wanted, the costs specific to using the different types of searches, and d or the maximal number of inverted list entries published for each term. Algorithm 5 details this process as follows:

Algorithm 5 Calculate-Tradeoff(Space, $Term_i \ldots Term_{num}$, Frequency)

1: Expect-Visit $\Leftarrow$ T / Frequency {Number of nodes Unstructured search will likely visit}
2: COSTS $\Leftarrow$ C_U*(Expect-Visit) - C_S *(Sending(query-terms))
3: **if** COSTS $>$ 0 **then**
4: RETURN 1 {pure unstructured search}
5: **else if** COSTS $<$ 0 AND Size($Term_i$) $<$ d **then**
6: RETURN -1 {pure structured search for this term}
7: **else**
8: space $\Leftarrow$ List($Term_i$) $\cap$ space
9: RETURN 1 {Use unstructured afterwards because of lack of more values}

First, the algorithm calculates the expected cost of conducting an unstructured search. The expected number of documents that will be visited in an unstructured search before finding T results is: T / term-frequency (line 1). For example, if we wish to find 20 results, and the frequency of the term(s) is 0.5, this search is expected to visit 40

documents before terminating. We can compare this value to that of using a structured search, whose cost is also known, and is proportional to the length of the inverted lists that need to be sent. We assume there is some cost, C_U associated with conducting an unstructured search on one peer. We also assume that some cost C_S is associated with sending one entry from the inverted list (line 2). Because the cost of unstructured search is C_U * T / Frequency, and the cost of structured search is bounded by C_S * ((max-1) * length($Term_i$) + T), the algorithm can compare the expected cost of both searches before deciding how to proceed (lines 3-6).

For many cases, a clear choice exists for which search algorithm to use. Let us assume that $C_U = C_S = 1$, and assume that all documents have been indexed, or $d{=}DOC - COUNTER$. When searching for common words, the cost of using the unstructured search is likely to be approximately T. Processing the same query with structured search will be approximately the number of documents ($DOC{-}COUNTER$) or a number much larger than T. Conversely, for infrequent terms, say with one term occurring only T times, the cost of an unstructured search will be $DOC - COUNTER$ or a number much larger than T, while the structured search will only cost a maximum of T * max-1 + T. Finally, structured search is also the clear choice for queries involving one term. Note that in these cases, no inverted lists need to be sent (max-1=0), and only the first T terms are returned. The cost of using unstructured search will be greater than this amount (except for the trivial case where the frequency of the term is 1.0).

There are two reasons why the most challenging cases involve queries with terms of medium frequency. In these cases, the cost of using both the structured and unstructured searches are likely to be similar. However, the expected frequency of terms is not necessarily equal to their

actual frequency. For example, while the words "new" and "york" may be relatively rare, the frequency of "new york" is likely to be higher than the product of both individual terms. As a result, the PHIRST approach is most likely to deviate from the optimal choice in these types of cases.

A second challenge results from the fact that we only published up to d instances of a given term. In cases where inverted lists were published without limitation, e.g. d equals N ($DOC - COUNTER$), the second algorithm contains only two possible outcomes – either the expected cost is larger for using structured search, or it is not. However, our assumption is that hardware limitations prevent storing this number of terms, and d must be set much lower than N. As a result, situations will arise where we would **like** to use inverted lists, but as these files have incomplete indices, this approach will fail in finding results in position d+ϵ. While other options may be possible, in these cases our algorithm (in lines 7-9) takes the d terms from the inverted lists, and conducts an unstructured search for all remaining terms. In general, we found this approach will be effective so long as the T $< d$, or the relationship, T $< d <<$ N exists. We further explore the impact of this limitation in the results section (5.7).

5.6 Dealing with Network Churn

The publishing and query algorithms address search issues related to storage hardware limitations, methods for equitably distributing inverted lists, and minimizing search cost. As we study a network of cellular phones, the system must additionally address temporary and permanent peer failures. This section provides a solution to this issue, known as churn, suitable for a distributed system.

5.6.1 The Churn Challenge

Churn can be defined as the turnover rate of nodes in the system over a given time period [19]. Based on previous work [19] we define churn (C) as follows: Given a sequence of changes in the set of N peer nodes, let U_i be the set of in-use nodes after the ith change, with U_0 the initial set. Churn is the sum over each event of the *fraction of the system that has changed state* in that event, normalized by run time t:

$$C \;=\; \frac{1}{t} \cdot \sum_{\text{events } i} \frac{|U_{i-1} \ominus U_i|}{N},$$

where $\ominus$ is the symmetric set difference.

Within cellular networks, most churn is likely to be caused by temporary changes of status where phones are momentarily not in service, but will eventually return into operation. For example a user might turn off her phone while sleeping, attend meetings, or go on vacations. We would expect real cellular networks to have a rather high churn rate due to these types of events. A less frequent type of churn is likely when a node decides to permanently change its status, say because a device breaks or when a user switches cellular carriers and receives a new number. While these events do not occur frequently, that nodes' information will be permanently lost if that device's data is not replicated.

Unstructured networks are not impacted by churn as long as the fluctuation of nodes' states still leaves the entire network with full connectivity [10]. As this method has no publishing or required routing of lookup information, it is unaffected by even very high churn rates. This has led some to claim that unstructured networks are most appropriate in these types of environments [10].

A variety of strategies have been proposed for dealing with churn in structured environments [19] to allow for continued connectivity once

a node fails. One classic approach is to **reactively** fix overlay network once a failure is detected. A second approach is to **periodically** check if nodes have failed. This can be done through constantly sampling neighbor nodes, and then preemptively replacing failed nodes before they cause a lookup failure. In environments with low churn levels the reactive methods perform better as they have no inherent communication overhead. However, in environments with even moderate churn levels, periodic methods perform significantly better [62].

It is important to differentiate between maintaining the connectivity of live nodes within the base DHT network, and the ability to find application data once stored in that network. Periodic approaches are better in handling churn, or finding nodes that are still participating in the network. However, our system must additionally maintain lost nodes' published inverted list information, something DHT's typically cannot do (refer back to the end of section 5.4). We now address how this challenge can be addressed.

5.6.2 Addressing Churn in a P2P Application

First, we present algorithm 6 to deal with planned types of disconnects. While we recognize that planned disconnects will not represent all churn events within a real system, this algorithm introduces the key elements of PHIRST. We then generalize this approach for dealing with unplanned churn events.

Note the strong similarity between this unpublishing algorithm, and the previously described publishing algorithm (algorithm 5.4). In both cases, we use a variable DOC-COUNTER of the global number of documents. However, in the unpublishing algorithm this value must be reduced. Similarly, the function Remove in line 8 and Update in

Algorithm 6 Unpublishing Algorithm(Document Doc)

1: Terms $\Leftarrow$ Preprocessed words in Doc
2: **for** $i = Terms_1$ to $Terms_{max}$ **do**
3: UNPUBLISH($Terms_i$, ID_{Source}, ID_{DEST})
4: **for** $j = File_1$ to $File_j$ **do**
5: COPY($File_j$, successor)
6: PUBLISH(DOC-COUNTER-1, ID_1)
7: **if** SIZE(ID_{DEST}, $Terms_i$) $<$ d **then**
8: REMOVE($Terms_i$, ID_{Source})
9: **else**
10: UPDATE($Terms_i$, COUNTER-1)

line 10 of this algorithm closely parallel lines 5-8 of the publishing algorithm. The purpose of the function Remove is to removes entry of the disconnecting peer, ID_{Source}, from the inverted list stored on peer ID_{DEST}. If the peer ID_{DEST} did not store the address of this peer, as it had already stored d entries of other peers, it simply reduces the counter of this term by one. Finally, lines 4–5 copies the inverted lists currently held by peer ID_{Source} to its successor within the DHT. From this point onward, this node will have the responsibility of responding to queries for these terms.

However, our assumption is that most churn effects result from temporary and unplanned failures where the failing node will not issue this unpublish command. In order to address this point, we present a solution where inverted list data are replicated to handle failures.

While the DHT's pointers need to be immediately updated in case of a node failure, however temporary, that node's data does not need to be copied if a set of backup copies exist. PHIRST relies on this set of backups with the assumption that the node's failure was temporary, and it will soon return to the network. This saves communication costs in copying inverted list data, assuming that node does soon rejoin the

network.

These data replicas can be easily created during the publishing stage. We modify the publishing algorithm (algorithm 5.4) to send its data to k peers instead of just one. We refer to a group of R redundant nodes, R_1, ..., R_k, where all nodes receive the same publishing data. Note that DHT's such as Chord [55] can allow for this functionality by sending a publishing message to the normal hashed peer ID_{DEST} within the Chord ring, and the next $k-1$ peers as well or, ID_{DEST}, ID_{DEST} + 1, ... ID_{DEST} + k - 1. For example, assume k=2, or two copies of each inverted list are stored. Referring back to Figure 5.1, all inverted lists that would normally be stored only on node 0, are now stored on nodes 0 and 1, items for 1 are stored on 1,3, etc. A temporary failure of node 1 will be handled by this node's successor, node 3 which also stored the same inverted list data. Join actions remain similar to those previously presented for DHT's with the exception that here a joining node must also be updated with the data currently being stored on the redundant nodes R_1, ..., R_k) that it is now joining. After this data is stored, the last node of the redundant set (R_k) can erase this data.

In a dynamic system, this approach will need to address two additional issues. First, care must be taken to address churn changes within the group of k redundant nodes after they are formed. For example, assume a new document is published, but node R_1 is temporarily unavailable. The data should still be published on the remaining k-1 nodes which now have the most updated version of the inverted lists. Once node R_1 becomes available again, it must receive the updated information. Second, we assume that most nodes become only temporarily unavailable, and thus a failure of a node, D, should not be a reason for immediately finding a new node to replicate the

data. However, after a certain time period, M, it must be assumed that node D has in fact left the system, and a new node must be found to store k nodes.

Algorithm 7 Replicating Data Under Churn(Node D)

```
1: Randomize start time
2: for Time = 1 to M step L do
3:    if Up(D) then
4:       Update(D)
5:       Break
6: Replace(D)
7: PUBLISH(DOC-COUNTER-NumDocs, ID_1)
```

Algorithm 7 outlines these two steps. The precondition for this algorithm is that a group of k nodes has already published the application data (inverted lists), and these nodes are aware of the others' existence. Within Chord this can be done through predecessor and successor pointers. Also, this algorithm is called once the redundant node set notices that a member of the set is not working (node D). We assume this is done through reactive sampling previously shown to be effective [19]. Next, we assume the remaining nodes mark what data has been published after node D failed, and can thus update node D once it becomes available.

Based on these preconditions, the algorithm operates as follows. Once a failure has been detected, the remaining nodes monitor the down node for a total time length of M (line 2 of the algorithm). Each node checks if the failed node resumes operation every L time units. Assuming they find it resumed functioning, it updates node D with the information it missed and the algorithm terminates (lines 3-6). One way to prevent two nodes checking at the same time, and enter a situation when both nodes attempt to update D simultaneously, is to

have each node start this algorithm only after a random start period (line 1). Assuming no two nodes picked the same number (which could be checked among the nodes calling this algorithm), this possibility can be averted. Finally, if the algorithm reaches line 8, we assume node D is permanently down. As Chord operates by searching successor nodes for information, this replacement node must the next node not currently in the set (R_1, ..., R_k), or node R_{k+1}. Additionally, we must reduce the number of documents in the system by the number of documents on node D. We represent this number as $NumDocs$ in line 9 of the algorithm.

However, comparing this approach to algorithm algorithm 6 reveals several challenges. In algorithm 6 each node could orderly remove all of its entries from the inverted lists. However, as the failures here are unpredicted, there is not possible in algorithm 7. Second, in line 9 of algorithm 7 we still refer to ID_1 as the node being responsible for holding the variable DOC-COUNTER. In fact, we now must store k copies of this variable (starting at node 1) to deal with churn. Finally, knowing the number of documents within node D is not trivial. We assume that nodes may contain different numbers of documents, and thus we may only assume some unknown quantity of x documents were on node D. In order to address this, we force every node within the k redundancy group to publish its number of documents to other members. However, we assume this approach it is infeasible for to maintain a list of all terms every other node has (as was done in lines 2-3 of algorithm 6) as this list will be quite large.

The value for k is a tunable parameter that must be set with care. As we deal with hardware with limited storage and assume communication is costly, care must be taken to not send data unnecessarily. However, sufficient servers must be present to make the probability

that all k peers will fail simultaneously extremely small.

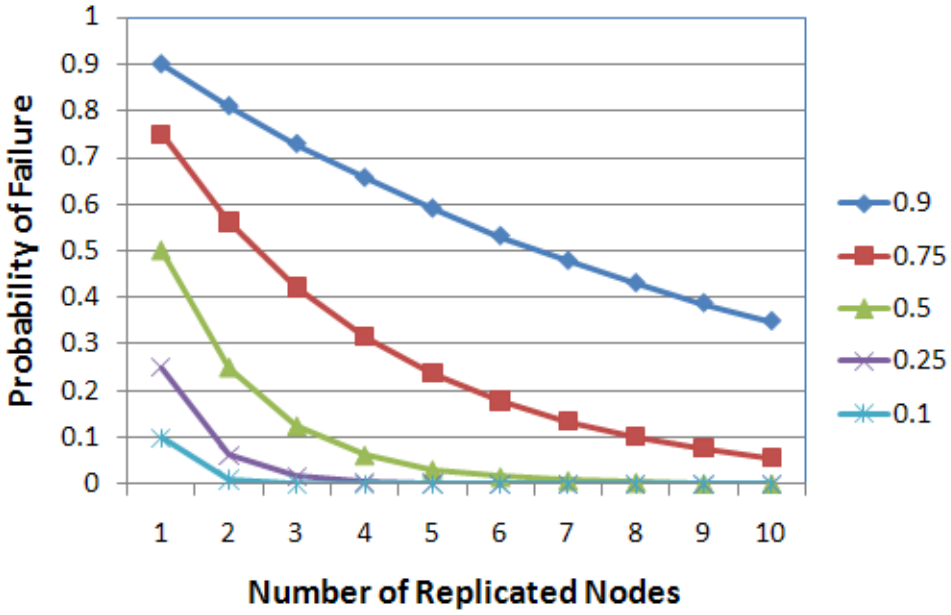

Figure 5.2: Calculating the probability all k nodes will fail.

Fortunately, the average system churn is typically a known or measurable quantity, and can be used to set the value of k. Assuming an average churn rate of p exists within the system, the probability the search algorithm will not find an inverted list given k redundant copies is p^k. Figure 5.2 graphically depicts the churn rates of 0.9, 0.75, 0.5, 0.25, and 0.1 with k being 1 – 10. For example, assume an average churn rate of 0.25, or 20% of the nodes will become unavailable in a given time period. The probability that at least one of two nodes in a redundant set will be available is 1-Probability(Failure), or 0.96. As the next section details, these probabilities are an effective guideline for setting k.

5.7 Experimental Results

In this section we present experimental results used to validate the effectiveness of the algorithms in this chapter. As our research goal

was to check if PHIRST is appropriate for medium sized newsgroups, we chose a corpus of 2000 real movie websites to conduct our experiments [58]. The results from the publishing experiments demonstrate that PHIRST actually becomes more feasible as more documents and agents are added to the network. We also created two types of query experiments. In one group we created artificial queries based on the frequency of words. This experiment demonstrated the theoretical strengths and weaknesses of PHIRST. We also studied real movie queries based on the Internet Movie Database (www.imdb.com). These experiments demonstrated that any weakness in PHIRST is likely to be insignificant in handling real queries.

5.7.1 Publishing Experiments

Recall that the publishing algorithm is based on storing a maximum of d entries in a given term's inverted list. We simulated the publishing process to study how this parameter affected the average number of stored inverted entries with and without term stemming. Figure 5.3 displays the average number of inverted terms (Y-axis) in groups of 50, 250, 500, 1000 and 2000 agents (X-axis). We assumed that every agent published 1 document taken from the movie corpus [58]. In the left graph, we used the Paice stemming algorithm [57] on each term before storing it. The right graph published each term without stemming. In both graphs we also ran the publishing algorithm with d=25 and 75.

Several interesting results can be seen from this graph. First, on average stemming saved approximately 50 words per document. This is because stemming lumps similar words, reducing the number of unique words occurring per document. Second, note the publishing algorithm has progressively larger storage savings as the number of nodes grows.

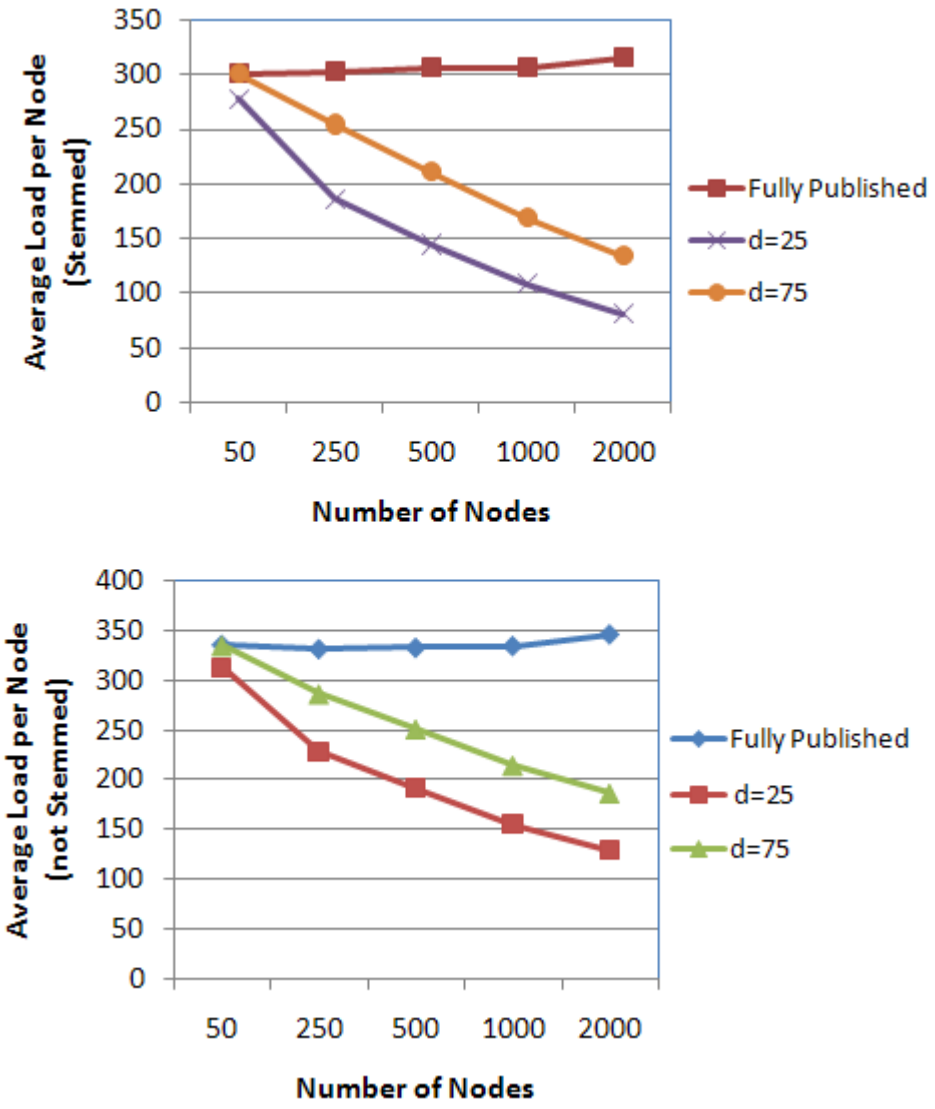

Figure 5.3: Comparing publishing requirements of full publishing versus publishing limited to d=75.

Assuming d=N, all terms will be stored, and no publishing gain will be realized by using the PHIRST approach. However, assuming d is kept fixed, the more documents that are added, the gap between d and N grows. This results in progressively more words exceeding the d threshold, and no longer needing to be stored. As a result, the publishing algorithm becomes **more** scalable the more nodes that are added, making full text search feasible even in very large P2P networks.

Finally, in this experiment we assumed each node had 1 document

Table 5.2: Average number of inverted entries if 1 document published for every 2 peers.

Number of Nodes	50	250	500	1000	2000
Fully Published	150.43	151.51	153.13	153.1265	157.8343
d=25	138.84	93.106	72.17	53.97	40.605
d=75	150.43	127.14	105.72	84.38	67.035

to publish. We also ran this approach with more dense (e.g. 2 documents per node) or more sparse (e.g. 1 document every 2 nodes) network assumptions. As one would expect, the number of terms each node stores is proportional to the total number of nodes. For example, Table 5.2 shows the sparse assumption of 1 document published for every two nodes. These values are identical to those in Figure 5.3 * 0.5.

We also found a Zipfian distribution of terms with a long tail of infrequent terms (see Figure 5.4). Similar distributions have been found in P2P systems for items such as file frequency [42, 43] and term frequency [33]. The storage saving results we found were from words with frequencies greater than d, or those terms towards the head of this distribution.

5.7.2 Query Experiments

We first conducted query experiments based on artificial queries chosen based on term frequency. Figure 5.4 displays the rank order of all words within the 2000 word corpus. We considered words of high frequency if they appeared in 30% or more of the documents. There were 200 words in this category. Note that high frequency words are not just "stop" words like "the", "and", or "a", but can be specific to the corpus. For example, these words included movie specific terms such as "character", "play", and "plot". At the other extreme, we

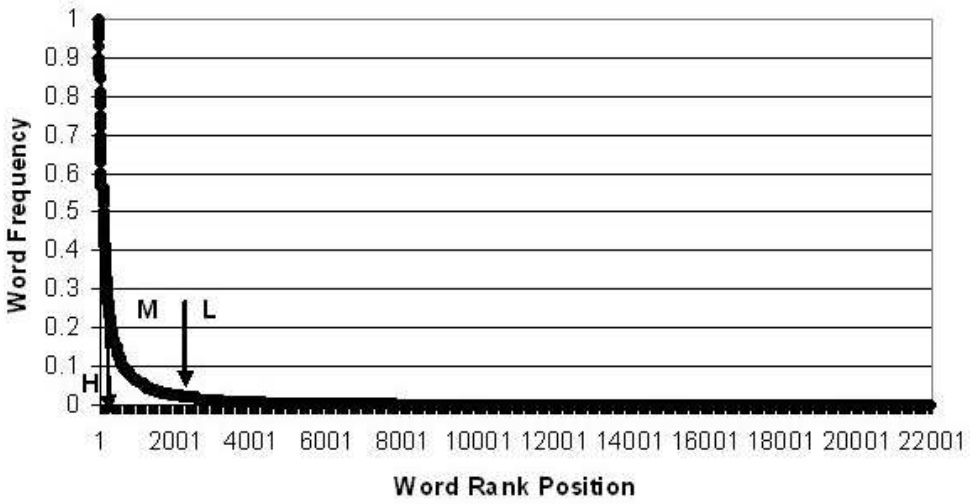

Figure 5.4: Distribution of words by rank order within a movie corpus.

define low frequency words as those appearing 50 times or less (frequency 2.5% or less). The large majority of terms were within this category due to the long tail of the term distribution. Finally, we assume medium frequency words are those between these extremes.

We created paired terms (2 terms) of all permutations of these categories. This involves words both with high frequency (HH), both of low frequency (LL), both of medium frequency (MM), low high combinations (LH), low medium combinations (LM), and medium high combinations (MH). Note that the order of the words does not impact the query algorithm as terms are first sorted by the query algorithm based on their frequency. For example, the low medium category (LM) is consequently equivalent to the medium low one (ML).

Next, we generated 1000 artificial queries from each category. We studied how many results were returned from each of 4 search algorithms. The Structured Search (**SS**) method published all terms and sent these indices between agents as necessary during queries. The Unstructured Search (**US**) used no publishing and used a random walk

Table 5.3: Comparing cost levels of SS, US, TTL, and PHIRST methods in LL, LM, LH, MM, MH, and HH artificial queries.

	SS	US	TTL=100	PHIRST
LL	1466	2000000	100000	1466
LM	2206	2000000	100000	2142
LH	3177	1987754	100000	2010
MM	20732	1865474	99953	13256
MH	60188	234211	95624	18075
HH	871986	19746	20077	19995

approach to find query results. The TTL=100 method used an unstructured search, but terminated after visiting 100 agents. Finally, the hybrid PHIRST approach implemented the publishing and query algorithms described in this chapter. In these experiments we used a value of d=75 in the PHIRST method.

Table 5.3 displays the average number of nodes visited (in the case of unstructured search) and / or the inverted list entries sent (for structured search) in finding 20 matches from each query (T=20). For simplicity, we assume that the costs of visiting nodes through unstructured search, and sending inverted list entries are equal, or $C_U = C_S$. As expected, we find that Structured Search (SS) is most expensive in finding common terms; where Unstructured Search (US) is most effective. Conversely, SS is most effective in finding rare terms. The hybrid PHIRST approach operates similarly to SS in finding rare terms (LL) and US in finding common items (HH). Note that in middle categories (for example MH) this approach sent the least amount of information. PHIRST saves costs by only sending a maximum of d entries even when structured search is deemed necessary. Furthermore, this approach switches between the SS and US methods as needed, saving additional costs.

We also studied the impact of the number of documents per node (document density) on these costs. The unstructured search (US) is most affected by the density of the documents. For example, assuming each agent stores 2 documents, the cost of using this search algorithm will be halved. Conversely, sparse networks make unstructured search less appealing. This tradeoff does come to light within the unstructured element of the Hybrid algorithm 5 in lines 1 and 2. However, unless extreme changes in the document density occur (e.g. every node containing a large percentage of the documents), differences in the search costs are so large that this parameter is unlikely to change which algorithm should be used in categories such as LL, LM, and LH. The structured approach is completely unaffected by document density, and thus the Hybrid's structured component is also unaffected as well.

The results in Table 5.4 display the number of query results returned from each search algorithm. This result underlies the potential strengths and weakness within the PHIRST method. Despite the lower costs of PHIRST, this approach was overall equally effective in returning the query results. When word combinations were frequent, the unstructured search component of the PHIRST method still found these results (thus MH was still successful). At the other extreme, assuming the word frequency of any term was less than d, at least one term was fully indexed. In these cases, complete recall was also guaranteed if structured search is used on the indexed term(s) followed by unstructured search to find all remaining terms. In these experiments, all terms taken from the L category were in less than d documents (e.g. L values had 50 or fewer instances while d=75), resulting in full recall for all of these categories (LL, LM, and LH) as well. As predicted in section 5.5, the query algorithm did have slight trouble in finding

Table 5.4: Comparing recall levels of SS, US, TTL, and PHIRST methods in LL, LM, LH, MM, MH, and HH artificial queries.

	SS	US	TTL=100	PHIRST
LL	3	3	0	3
LM	68	68	2	68
LH	1167	1167	47	1167
MM	874	874	93	870
MH	4626	4626	1180	4626
HH	5000	5000	4997	5000

series of terms of medium frequency. Note that the PHIRST method did return slightly fewer results in the MM case (870 versus 874).

We found that this limitation was negligible in answering real world queries once d was significantly higher than T. To verify this claim we used the 1000 most popular real movie keywords taken from the Internet Movie Database Internet Movie Database[1] taken from October 25, 2006. These queries were typically between 1 and 4 words (mean 1.94).

Table 5.5 compares the number of results found from these queries with SS, US, and TTL=100 methods, and the PHIRST method with d=75 with variable values for T. Note that the PHIRST algorithm found nearly all results (99.89%) when only 5 results were requested (T=5). PHIRST held up fairly well even when 20 matches (T=20) were required with 97.78% of all matches found. The recall of the PHIRST approach dropped with T (92.77% at T=50, and only 33.23% at T=N). This confirms the claim that in real queries the recall of the PHIRST approach will be nearly 100% for T $<<$ d (e.g., T=5), but performed poorly once T $>>$ d (e.g., T=N).

Table 5.6 displays the search costs for finding these real queries

[1](http://www.imdb.com/Search/keywords)

Table 5.5: Comparing recall levels of SS, US, TTL, and PHIRST methods with regard to different numbers of results (T).

	SS	US	TTL=100	PHIRST
T=5	4592	4592	2138	4587
T=20	15598	15598	3712	15252
T=50	30347	30347	4534	28154
T=2000	105649	105649	5254	35087

Table 5.6: Comparing cost levels of SS, US, TTL, and PHIRST methods with regard to different numbers of results (T).

	SS	US	TTL=100	PHIRST
T=5	57680	591841	86578	12006
T=20	68696	1181515	97735	24976
T=50	83435	1567039	99269	38744
T=2000	158737	2000000	100000	68610

for the 4 algorithms described in this chapter assuming $C_S = C_U = 1$, and each agent stored only one document. We again found the PHIRST approach had significantly lower search costs that all three of the other approaches. Again, observe that the advantage to the PHIRST approach is most effective when d>>T. If T=5, the PHIRST approach has nearly 1/5 the cost of the next best method (SS) (with a high recall of 99.89%). If T=20, its cost is still nearly 1/3 that of the next best method (SS) (recall still high at 97.78%). If T=N, the cost advantage of the PHIRST approach is under 1/2 from the next best method (TTL=100) (recall only 33.23%).

5.7.3 Churn Experiments

Recall that the PHIRST approach to handling churn requires that k copies of each inverted list must be stored. We conducted experiments studying the relationship between this value of k, the system's pub-

Table 5.7: Comparing the impact of redundant nodes on publishing load, query results, and search cost within the Hybrid method with d=75 and T=20 when node failure results in search termination.

Replicated Nodes (k)	Publishing Load	Query Results	Search Cost
1	134.07	6756	6015
5	670.35	14692	25308
15	2011.05	15252	24969

lishing requirements, and the probability inverted lists were available. The goal was to achieve at least 95 percent of the query results when confronted with churn compared to the results achieved when no churn existed.

We simulated conditions with k being 1, 5, and 15 and studied their impact on the publishing storage and query results within the Hybrid algorithm. We revisited the real-world queries from the previous dataset, where the goal was to return 20 matches (T), and a scenario with a very high churn rate of 0.5. To simulate churn, we created random snapshots of the simulator where half of the nodes were chosen at random to have failed with uniform distribution. In the first set of experiments, we assumed that when the entire set of k nodes were "down" the query would fail. The results from this experiment are found in Table 5.7.

As these results indicate, there is a clear tradeoff between having additional nodes within k, their storage requirements, and the query results. Setting k=1 has the lowest publishing cost, but also results in losing many of the query results. Setting k=15 allows for finding all possible results, but the highest cost. As predicted from Figure 5.2, setting k=5 will result in an effective tradeoff between achieving over 95% recall from the inverted lists (0.96% of the results from k=15) while still keeping publishing costs relatively low (1/3 of the publishing

Table 5.8: Comparing the impact of redundant nodes on publishing load, query results, and search cost within the Hybrid method with d=75 and T=20 when node failure results in using unstructured search.

Replicated Nodes (k)	Publishing Load	Query Results	Search Cost
1	134.07	15428	566477
5	670.35	15259	59994
15	2011.05	15252	24969

costs of k=15).

In the above results, we assumed a query would fail if structured search was desired, but the node with the inverted list(s) had failed. Next, we repeated the above experiment, but assumed unstructured search would instead be used if the node with the inverted list failed. These results are found in Table 5.8. As one would expect, when k=15 the entire k set of nodes never failed, and thus random search was never used. Thus, the results here mimicking those in Table 5.7. Conversely, setting k=1 resulted in often using the random search. Note that these results even outperform those of the base Hybrid approach, as the Hybrid approach performs unstructured search for approximately 50% of queries. However, this came at a very high search cost. Finally, k=5 again provides a good tradeoff between these factors.

Based on these results we concluded that the PHIRST approach was successful in reducing publishing costs, even in systems with a very high churn rate.

5.8 Conclusion and Future Work

In this work we present a hybrid P2P search approach that leverages the strengths of structured and unstructured search. We present a P2P publishing algorithm that insures that no peer can hold more

than d entries in its inverted file of a given term. This ensures that no one peer is required to hold disproportional amounts of data. Our approach is highly scalable in that every peer typically stores fewer entries as the number of peers grows. This allows us to partially index all words in the corpus while keeping storage costs low. We also present a querying algorithm that select the best search approach based on global frequencies of all words in the corpus. This allows us to select the best method based on estimated cost. Our approach uses unstructured search to compensate for the lack of published inverted file locations of frequent terms and structured search to location rare terms. Finally, we present a distributed approach for handling churn, where k copies of each inverted index are stored. We found that the value for k can be easily set to match the average churn rate within the system.

In the future, we hope to deploy our system in a moderate sized group of cellphones participating in a P2P network. We intend on measuring the actual costs involved with deploying our approach in DHT implementations such as Chord [55] and random walk approaches [74] on live systems instead of using the simulated costs we used for the experiments in this work.

Chapter 6

Final Remarks

In this thesis, we explored how an algorithm selection approach could be used to create adaptive coordination methods in a several domains. In the first chapter, we studying how to create adaptive methods for homogeneous robots without communication. In the second chapter, we present two types of adaptive communication methods, uniform and neighborhood. In the third chapter, we studied how adaptive methods could be applied to optimization domains, such as scheduling and graph coloring problems. In the fourth chapter, we present an method for switching between structured and unstructured full-text search algorithms in Peer to Peer networks.

In all domains, the key to the success of the adaptive approaches we present was identifying what attributes differentiate between coordination algorithms. In the robotic domains this was done through the CCC measure we presented. In the schedule optimization domain, we developed a tightness measure to quantify the expected impact of sharing information. In the Peer to Peer search domain, a global measure of term distribution allowed us to equate between search algorithms. In all domains, once these attributed were found, we successfully created adaptive methods that significantly outperformed the static methods they were based on.

In addition to the domain specific future directions presented at the end of each chapter, we hope to pursue several more general questions. One key goal is to develop a formalized model that includes all of the coordination measures we present. We hope to study if the phase transition model that has often been applied to distributed constraint optimization problems can be extended to coordination problems in general or to the specific subset of domains we studied. If such a model can be developed, it may be able to predict which coordination method will perform best, or to predict a-prior what the productivity level of a given method will be without running any trials.

Another key question focuses on the optimality of the adaptive methods we created. As we described in the introduction, finding **the** optimal coordination policy is an intractable problem for most real-world domains. Thus, our approach focused on finding methods of improving productivity beyond the highest levels static methods were capable of. While the methods we created did achieve statistically significant improvements, we hope to study if it is possible to quantify how good an approximation of optimal behavior was achieved. Specifically, we hope to study if upper and lower bounds can be computed within the adaptive methods we present relative to the theoretical optimal behavior.

Finally, the basic approach of this thesis was to select the best coordination algorithm from a group of known algorithms. In the future, we hope to apply the lessons from this thesis to create new coordination algorithms in related problems such task allocation, coalition formation, and large scale teams. We are hopeful that this work will lead to additional advances in multiagent systems.

Bibliography

[1] www.google.com.

[2] J. A. Allen and S. Minton. Selecting the right heuristic algorithm: Runtime performance predictors. In *Canadian Conference on AI*, pages 41–53, 1996.

[3] R. Arkin and T. Balch. Cooperative multiagent robotic systems. In *David Kortenkamp, R.P. Bonasso, and R. Murphy, editors, Artificial Intelligence and Mobile Robots. MIT/AAAI Press, Cambridge, MA.*, 1998.

[4] T. Balch. The impact of diversity on performance in multi-robot foraging. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 92–99, Seattle, WA, USA, 1999. ACM Press.

[5] T. Balch. www.teambots.org, 2000.

[6] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[7] T. R. Balch and R. C. Arkin. *Behavioral diversity in learning robot teams.* PhD thesis, 1998.

[8] S. L. Brue. Retrospectives: The law of diminishing returns. *The Journal of Economic Perspectives*, 7(3):185–192, 1993.

[9] S. Brueckner and H. V. D. Parunak. Information-driven phase changes in multi-agent coordination. In *Engineering Self-Organising Systems*, pages 104–119, 2005.

[10] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[11] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[12] J. Davin and P. J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1057–1063, New York, NY, USA, 2005.

[13] G. Dudek, M. Jenkin, and E. Milios. A taxonomy for multi-agent robotics. *Robot Teams: From Diversity to Polymorphism, Balch, T. and Parker, L.E., eds., Natick, MA: A K Peters*, 3:3–22, 2002.

[14] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3:375–397, 1996.

[15] E. H. Durfee. Practically coordinating. *AI Magazine*, 20(1):99–116, 1999.

[16] M. Y. J. O. Eiichi Yoshida, Tamio Arai. Local communication of multiple mobile robots: Design of optimal communication area for cooperative tasks. *Journal of Robotic Systems*, 15(7):407–427, 1998.

[17] C. B. Excelente-Toledo and N. R. Jennings. The dynamic selection of coordination mechanisms. *Autonomous Agents and Multi-Agent Systems*, 9:55–85, 2004.

[18] M. Fontan and M. Matarić. Territorial multi-robot task division. *IEEE Transactions of Robotics and Automation, 14(5), 1998.*, pages 815–822, 1998.

[19] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–158, New York, NY, USA, 2006. ACM Press.

[20] D. Goldberg and M. Matarić. Interference as a tool for designing and evaluating multi-robot controllers. In *AAAI/IAAI*, pages 637–642, 1997.

[21] D. Goldberg and M. Matarić. Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In *Robot Teams: From Diversity to Polymorphism*, pages 315–344, 2001.

[22] C. Gomes, W. van Hoeve, and B. Selman. Constraint Programming for Distributed Planning and Scheduling. In *AAAI Spring Symposium on Distributed Plan and Schedule Management*, 2006.

[23] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence (AIJ)*, 126(1-2):43–62, 2001.

[24] L. Gravano, H. García-Molina, and A. Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.

[25] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[26] L. Hogg and N. Jennings. Socially intelligent reasoning for autonomous agents. *IEEE Transactions on Systems, Man and Cybernetics - Part A*, 31(5):381–399, 2001.

[27] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artif. Life*, 5(2):173–202, 1999.

[28] B. Horling, R. Mailler, and V. Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.

[29] M. Jager and B. Nebel. Dynamic decentralized area partitioning for cooperating cleaning robots. In *ICRA 2002*, pages 3577–3582.

[30] M. Jager and B. Nebel. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *IROS*, pages 1213–1219, 2001.

[31] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[32] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[33] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in dht-based peer-to-peer networks. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 339–348, Washington, DC, USA, 2005. IEEE Computer Society.

[34] G. A. Kaminka and R. Glick. Towards robust multi-robot formations. *ICRA-06*, 2006.

[35] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[36] A. M. I. R. E. Kaplansky and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, July 2002.

[37] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. *AAAI*, pages 611–616, July 2004.

[38] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Proc. 17th International Conf. on Machine Learning Morgan Kaufmann, San Francisco, CA*, pages 511–518, 2000.

[39] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

[40] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *CP2003*, pages 899–903, 2003.

[41] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems*, 2003.

[42] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing p2p file-sharing with an internet-scale query processor, 2004.

[43] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid p2p search infrastructure, 2004.

[44] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM Press.

[45] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *National Conference on Artificial Intelligence*, pages 768–773, 1991.

[46] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking dcop to the real world: Efficient complete

solutions for distributed multi-event scheduling. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.

[47] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.

[48] R. Mailler and V. Lesser. Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. In *AAMAS '04*, pages 446–453, New York, 2004.

[49] T. W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[50] M. Matarić. Reinforcement learning in the multi-robot domain. In *Autonomous Robots*, pages 73–83, 1997.

[51] M. J. Matarić. Using communication to reduce locality in multi-robot learning. In *AAAI/IAAI*, pages 643–648, 1997.

[52] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[53] P. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence (AIJ)*, 161:149–180, 2005.

[54] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic "phase transitions". *Nature*, 400(6740):133–137, 1999.

[55] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, San Diego, CA, September 2001.

[56] E. Ostergaard, G. Sukhatme, and M. Matarić. Emergent bucket brigading - a simple mechanism for improving performance in multi-robot constrainedspace foraging tasks. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 29–30, 2001.

[57] C. D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, 1990.

[58] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *EMNLP '02: Proceedings of the ACL-02*, pages 79–86, 2002.

[59] H. V. D. Parunak, S. Brueckner, J. Sauter, and R. Savit. Effort profiles in multi-agent resource allocation. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 248–255, New York, NY, USA, 2002. ACM Press.

[60] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

[61] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, pages 21–40, 2003.

[62] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[63] J. R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 118–165, 1976.

[64] A. Rosenfeld, G. Kaminka, and S. Kraus. Adaptive robot coordination using interference metrics. In *The Sixteenth European Conference on Artificial Intelligence*, pages 910–916, August 2004.

[65] P. Rybski, A. Larson, M. Lindahl, and M. Gini. Performance evaluation of multiple robots in a search and retrieval task, In Workshop on Artificial Intelligence and Manufacturing, 1998.

[66] M. Schneider-Fontan and M. Matarić. A study of territoriality: The role of critical mass in adaptive task division. *In Maes, P., Matarić, M., Meyer, J.-A., Pollack, J., and Wilson, S., editors, From Animals to Animats IV, pages 553–561. MIT Press.*, 1996.

[67] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, Seattle, WA, 1994.

[68] R. S. Sutton. Reinforcement learning: Past, present and future. In *SEAL*, pages 195–197, 1998.

[69] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[70] A. Tews. Adaptive multi-robot coordination for highly dynamic environments. In *CIMCA*, 2001.

[71] R. Vaughan, K. Støy, G. Sukhatme, and M. Matarić. Go ahead, make my day: robot conflict resolution by aggressive competition. In *Proceedings of the 6th int. conf. on the Simulation of Adaptive Behavior*, 2000.

[72] C. J. C. H. Watkins. Learning from delayed rewards. *Ph.D. Dissertation, Kings College*, 1989.

[73] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[74] Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *IEEE INFOCOM*, 2006.

[75] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

[76] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In V. Lesser, editor, *ICMAS*, pages 401–408. MIT Press, 1996.

[77] K. Yoon and C. Hwang. *Multiple attribute decision making: an introduction.* Prentice Hall, Thousand Oaks: Sage, 1995.

[78] W. Zhang. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In *Principles and Practice of Constraint Programming*, pages 153–167, 2001.

[79] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale over-

lay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[80] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.