COOPERATIVE CONTROL OF SYSTEMS WITH VARIABLE NETWORK TOPOLOGIES

A Dissertation
Presented to
The Academic Faculty

by

William Grant Whittington

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
August 2013

COOPERATIVE CONTROL OF SYSTEMS WITH VARIABLE NETWORK TOPOLOGIES

Approved by:

Prof. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Jeffrey Sitterle
Georgia Tech Research Institute
Georgia Institute of Technology

Dr. Vitali Volovoi
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: June 27, 2013

ACKNOWLEDGEMENTS

The PhD process has been one of the most rewarding and frustrating journeys of my life. What attracted me to math and science initially was the desire for understanding and the ability to solve problems in a methodical process. Each problem had a concrete solution and with sufficient understanding that solution can be found. Of course graduate school showed me this is only true for the most well understood problems. The frontiers of science are ripe for new ideas and discoveries, but they will likely be hard fought to gain. The unknown nature of research makes progress hard to schedule; some days moving forward, some days realizing an idea is not as good as it sounded weeks ago and some days exploring a beneficial idea that was not considered earlier. The nature moving of two steps forward, one step back, one step to the side before moving forward again has taught me as much humility and patience as it has about engineering.

I would like to first thank my committee for their help and guidance. Each person brought something different to the technical side of this research, and it has certainly benefited. This research brought many different ideas together and it was always useful to be directed to a paper or idea that I had not considered before. One of the most helpful lessons learned from the committee was to look at the problem from a larger scale, and not focus solely of the color of the trees rather than the forest as a whole.

Dr Mavris, you took a headstrong a young man and molded him into a well rounded engineer, with a needed dose of humility. I will leave graduate school a better presenter, critical thinker and listener than when I came in. I would like to think that I always had the smarts to do well in school, but you helped me gain all the other skills I would need for the working world.

Dr Sitterle, you gave me an opportunity to pursue this research that I would not have had otherwise. You took a chance on me and I will forever be grateful for that. You pointed person who loved to solve problems but lacked some direction and pointed me the right way. Your even keeled guidance has always helped me remain grounded and focus on my work in a neutral manor. Every meeting we had left me with a new idea, guidance and/or more eager to get back to my work. I cannot think of another person I would rather come to for guidance, and your help has been valuable.

Dr Chen, your final research project helped me become a better researcher. Your assignment to find a peer reviewed article with mistakes and correct them at first seemed pompous to me. I quickly realized you wanted us to gain the skill of reading other peoples work with a critical eye and not take the results as gospel. This helped me greatly in the PhD process.

I would like to thank my colleagues Bryan Bolling, Kevin Johnson and Kemp Kernstein. Probably the luckiest part of graduate school was being interested in the same classes as Kevin and Bryan. Having classes with you guys was a pleasure. Kemp, I think we are kindred spirits in our love of solving interesting problems. Every time you came by my desk to see what I thought of an idea, I was glad for the opportunity.

I would like to thank everyone I worked with in grad school, but especially Peter Suh and Shawn Johnson. Peter, you shared a trenches mentality to research that not many others do and were willing to spend long hours doing the dirty work with me. I always enjoyed our talks about esoteric modeling techniques and just hanging out. Shawn, you were probably the best person I could have worked with. Always willing to do hard work and looking at avenues that normally I would not. Your mellow nature also helped me be more composed when I got worked up over one issue or another. Rebecca Douglas, we butted heads sometimes but I am glad I worked for you. It was difficult to earn your respect, but once I did you gave me many opportunities that others would not have. The trust and freedom you allowed me to pursue my ideas was greatly appreciated. Not many people would have allowed a student barely in his second year to present in front of Joint Forces Command and some higher ups are Raytheon.

Stephanie Large, you know how important you are to me and this process is no exception. Thank you for your support.

Finally I would like to thank my friends and family. Mark and Brad, you have been my best friends for most of my life. Being able to call or visit you has always been valuable to me. To my family: you have supported me in every venture in my life and I cannot thank you enough.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS AND ACRONYMS

CAS	Complex Adaptive Systems
UAV	Unmanned Aerial Vehicle
SIS	Susceptible Infected Susceptible
SIR	Susceptible Infected Resistant
LAP	Linear Assignment Problem
LSAP	Linear Sum Assignment Problem
QAP	Quadratic Assignment Problem
MILP	Mixed Integer Linear Programs
NAP	Nonlinear Assignment Problem
MIAP	Multi Index Assignment Problem
3IAP	3 Index Assignment Problem
A3IAP	Axial 3 Index Assignment Problem
P3IAP	Planar 3 Index Assignment Problem
GRASP	Greedy Randomized Adaptive Search Procedure

LIST OF SYMBOLS

$\otimes$	Kronecker Product
L(G)	Laplacian matrix of graph G
A(G)	Adjacency matrix of graph G
I_n	Identity matrix of size n x n
N	Number of nodes in a graph
n	Number of links in a graph

SUMMARY

Automation has become increasingly prevalent in all forms of society. Activities that are too difficult for a human or to dangerous can be done by machines which do not share those downsides. In addition, tasks can be scheduled more precisely and accurately. Increases in the autonomy have allowed for a new level of tasks which are completed by teams of automated agents rather than a single one. This has many benefits; teams of machines can perform objectives greater than the sum of its parts. With the proper strategy, teams of less capable agents can complete the same task as a more capable single agent, while doing so for less cost and more robustness to failures. Teams of agents can also complete tasks that would be impossible for a single system by dividing goals amongst the members and coordinating their completion.

This benefit does not come without some downside. Programming single automated systems to do easy tasks can be difficult, making teams of them act in unison for some shared objective adds to this difficulty. Cooperative control is the study of strategies and techniques by which teams of automated agents can complete some shared goal. It is the study of this field which will be the main focus of this research. The difficulty of cooperative control leads to the majority of research in the field being highly focused on individual problems: how can a team of UAVs best fly in formation or how can a system of robots best explore and forage in an unknown area?

One of the most common assumptions made in these problems is about the communication between the individual members of the team as well as the variation in the problem. Often full knowledge of the entire system is assumed, or at least that the communication network is not changing or changing while meeting strict requirements. Coordination is very important in these problems and ways to get the individuals to have agreement is a main area of study. Making these assumptions helps alleviate this problem. However in real life systems failures may occur in individual agents, and if the system is not prepared for them may compromise the overall objective. In a similar fashion, the variation of actors, tasks they need to perform and their ability to perform those tasks also makes the problem more difficult. It is the goal of this research to look at cooperative control methods which can operate under sub-optimal communication networks in which changes may happen often as well as failures while still completing the overall mission. Such a system would be more robust and resilient, but the disagreement of information in this sub-optimal communication scheme is a difficult problem to solve.

Creating a high level control scheme to enact proper strategies in a cooperative system is the goal of this research. The control must be able to work in any network configuration as well as various problem scenarios. Adjusting the system to have desired performance in bad cases will likely decrease performance in situations when that extra level of effort is not needed. In order to alleviate this problem the control system should be adaptive based on its communication network. Determining how it will do so is another goal of this research.

Finally, this cooperative control scheme will be evaluated via a test bed designed to reproduce a wide range of communication networks, problem types and system changes in impact and frequency.

CHAPTER 1: INTRODUCTION AND MOTIVATION

Automation has become increasingly prevalent in many aspects of society, ranging from house-hold cleaning and maintenance to manufacturing to military operations. The ability of machines to do tasks which are undesirable or dangerous and do that task consistently and reliably lends itself well to many existing applications and opens avenues for future applications previously limited by human ability. The difficulty of using automation in this sense is the need to make sure the agent can act and be controlled in such a way to consistently and effectively fulfill its task. It has been found in many cases that teams of autonomous agents acting in a coordinated fashion can do the same job cheaper, quicker or with more ease than a single more capable but also more expensive agent. Such teams are also more reliable and robust to errors or failures simply because the teamwork aspect can fill in gaps that may arise from these problems. A single failure can eliminate a single autonomous agent, but teams of agents may be able to remain operational with similar failures. As such, the study of autonomous teams has become popular in recent years in order to tackle the inherent problems of communication and coordination in addition to the automation of each single agent. This field is known as cooperative control, and it is within this field that the thrust of this research is focused. Specifically robustness and reliability of the system will be analyzed, with the goal of having overall system success despite a high level of problem variability and possible singular failures within the system. In addition, the agents of the team will operate with limited resources, and using these resources in an effective way is of importance.

Cooperative control can be boiled down into two basic components: determining the desired behaviors and actions given a set of system parameters, and the communication or sharing of information between the agents. Individual and group behavior of the system is often highly specialized for the specific task and usually does not carry over to other tasks. Communication however is much more applicable over a range of cooperative control problems. For specific tasks and agent behaviors communication can be ignored due to the highly regular and well known system it is operating in. For many tasks some sort of information sharing is required to ensure that the agents are in fact cooperating and not merely acting as individuals in a group. Even systems without communication often need some sort of sensor information about its fellow teammates, such as formation flying of UAVs [47] and Multi-Robot Remote Driving [51] .

The effect of what is communicated or how depends greatly on the system and the goal to be completed. It is desirable to have as complete an idea of the system as possible and that information is as current as possible. This would correspond to a fully connected communication network and rapid communication. Much research on cooperative control makes the assumption that all information is known by all agents or that the associated communication network is fully connected [36] [120] [121] . These two statements do not necessarily mean the same thing depending on the nature of communication, but in general these assumptions are as good as can be expected in real life situations. Other research not making this assumption generally assumes the networks are static or only variable to a limited degree. While more difficult, the unchanging nature removes some uncertainties which may exist in real networks. These assumptions make the problems much easier to solve, but also greatly limit the flexibility, robustness and reliability of the system. Existing techniques which make strict requirements on the communication network are not assured to work when such variability exists within the system.

By creating a control scheme which requires perfect knowledge of the system, it is not certain that control will succeed if a failure occurs. Communication failures would at the very least ensure some information is old and at worst separate agent(s) from the rest of the system. This can be a major issue. Some research has been conducted to analyze cooperative control with variable communication networks. Olfat-Saber et al [108] analyzed specific communication methods and techniques in a variable network, but it is mainly concerned with the convergence of system data and is somewhat limited in some cooperative control problems. One of the main concerns is the acceptable level of agreement of information. When using a GPS for driving directions, an error of ten feet is probably acceptable, for close proximity formation flying, that sort of positional error might be too high for some maneuvers. Clauset et al [35] looked into control in a more general sense, assuming the communication network was variable, but did not arrive at hard answers or conclusions. Rather, key features of such systems were qualitatively discussed and in the end it was left to other authors to analyze such systems in more depth. It is difficult to apply cooperative control effectively even in the best of situations communication wise, which is the reason why most of the research assumes communication networks are compact. In fact, some practical applications of cooperative control will go to great means to ensure the communication network is as compact as possible, sacrificing some mission variability to do so [144] . It could be said that the majority of concrete research into cooperative control seeks to answer the following question: given a system of agents with a fixed communication network, what is the best way to achieve a set task or tasks? That is to say how can control be conducted *to or on* the said network [35] ? However, the goal of this research is to look at the problem from a different perspective: Regardless of what the network is, if and how can cooperative control be applied to complete a desired task? How can other system changes be accounted for without failures of the overall mission? Before going into more detail of the problem, a brief notional example of some of these issues will be presented below. How and why existing techniques are not suited to such a problem are presented.

1.1 Notional Motivating Problem

In order to demonstrate some of the needs for this research, a brief notional example will be presented. The scenario consists of a team of UAVs whose main mission is to jam a set of enemy communication locations. The ability of each UAV to complete the jamming process is based on its transmit power, which is fixed for each platform. It is desired that the maximum amount of jamming signal strength is focused on all targets, to ensure the success of the jamming mission. The ability to jam multiple targets at once does exist, but comes at the cost of splitting the available transmit power of each platform.

An additional goal in this problem is that the jamming of each platform is continuous without any gaps. This is desired because it will effectively limit all communication from each target; even a minor gap in jamming will allow the target to send out some communication before jamming is continued. Enemy signals alerting the presence of allied platforms or launch signal initiations are such examples of cases when even momentary gaps in jamming to cause system failure.

The behavior of the system is defined by the choices each platform makes on where/how to apply its jamming. This can be thought of the strategy each UAV employs. A successful cooperation scheme would ensure that the choice of jamming distribution is sufficient to give constant coverage of all targets.   Ideally, the system would never encounter changes and once the targets are effectively distributed between the UAVs, the only need is to maintain this configuration until the mission is over. Such a configuration is given below in Figure 1. It is made up of a different number of UAVs and targets to demonstrate that each UAV has the ability to jam multiple targets, albeit at a reduced effectiveness compared to jamming a single target.

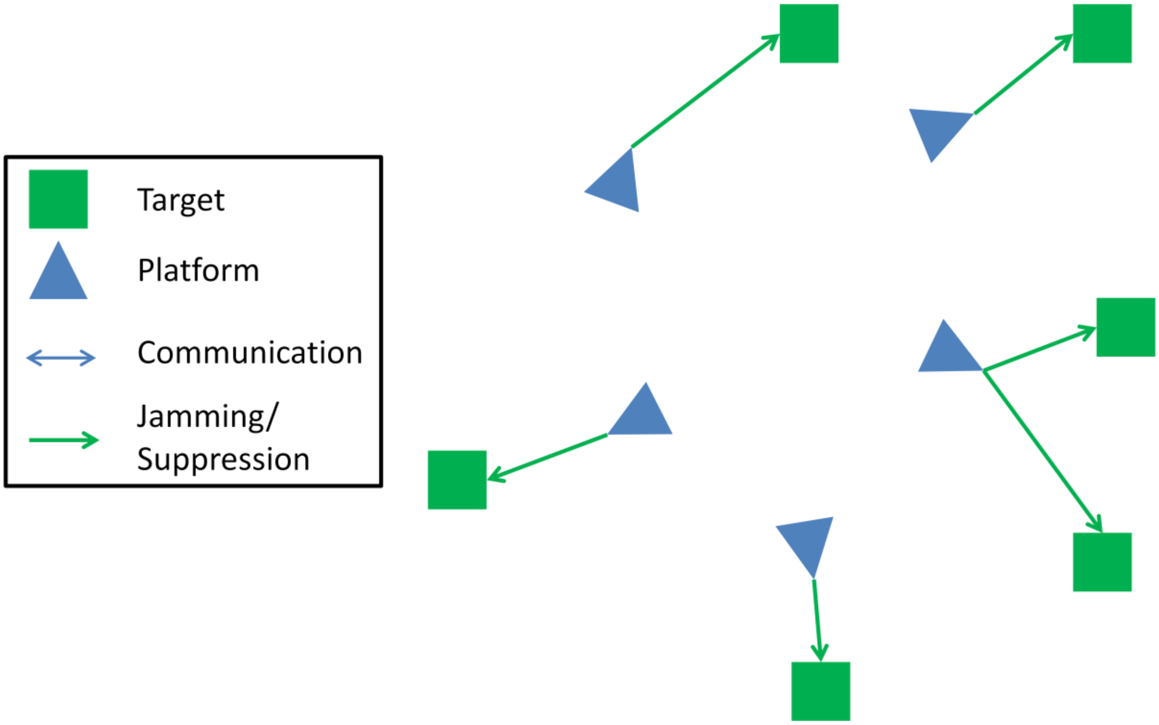


Figure 1 Notional Distribution of Applied Jamming in a Suppression Mission

However system changes may occur, and when they do other specialized behavior needs to be implemented to ensure constant coverage. The difficulty of the problem stems from the fact that changes may be known only by a subset of the total number UAVs when it happens. It takes some amount of time for information to travel to the rest of the team. This disagreement of system information is problematic and needs to be addressed in some way. One possible solution to this information lag would be to make the UAV communication network as compact as possible. That corresponds to having a direct line of communication between each pair of UAVs, meaning that each UAV can communicate directly with all other UAVs. Such a network is said to be fully connected, because each possible node pair is linked. An example of such a system is given below in Figure 2. Such a system would be more resilient to some changes in the system. For example, if a given UAV temporarily loses its jamming ability, this could be messaged to the rest of the team immediately, and a new configuration implemented to ensure constant coverage. Such an occurrence is shown below in Figure 3.

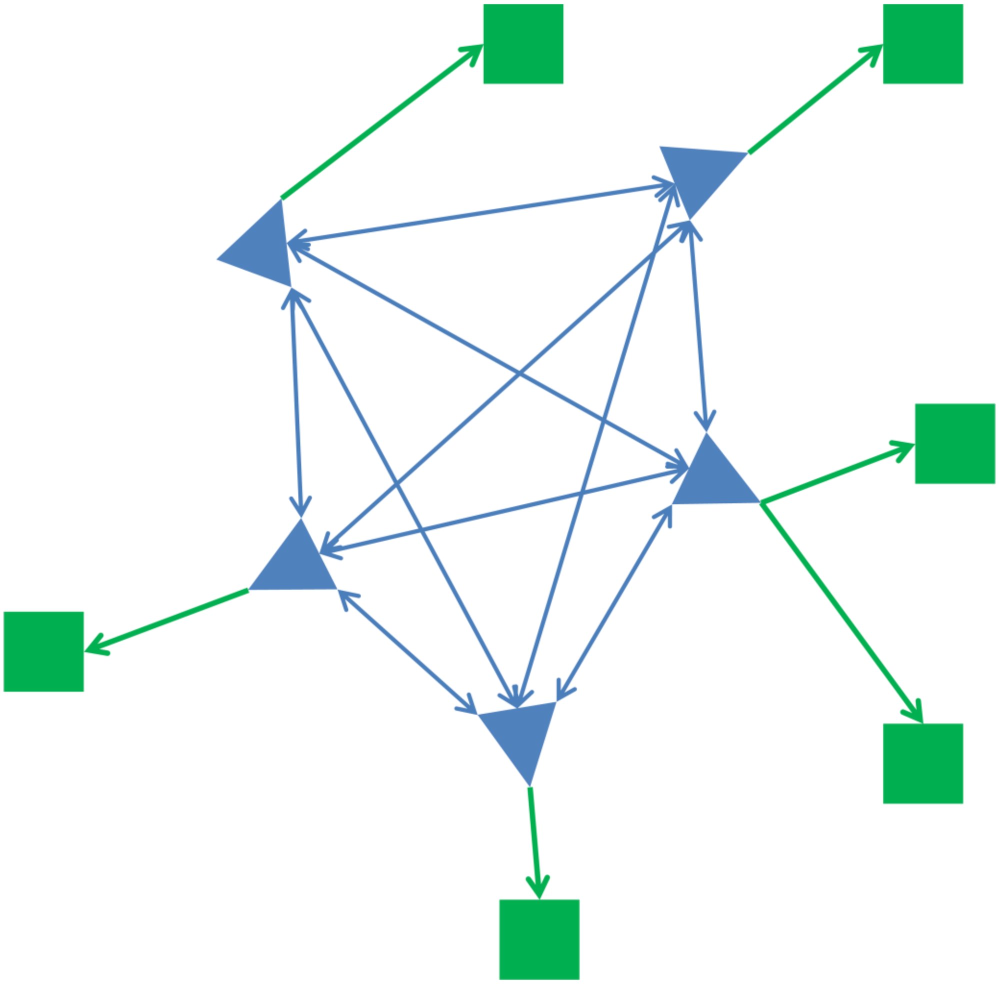


Figure 2 Example Configuration with Fully Connected Network

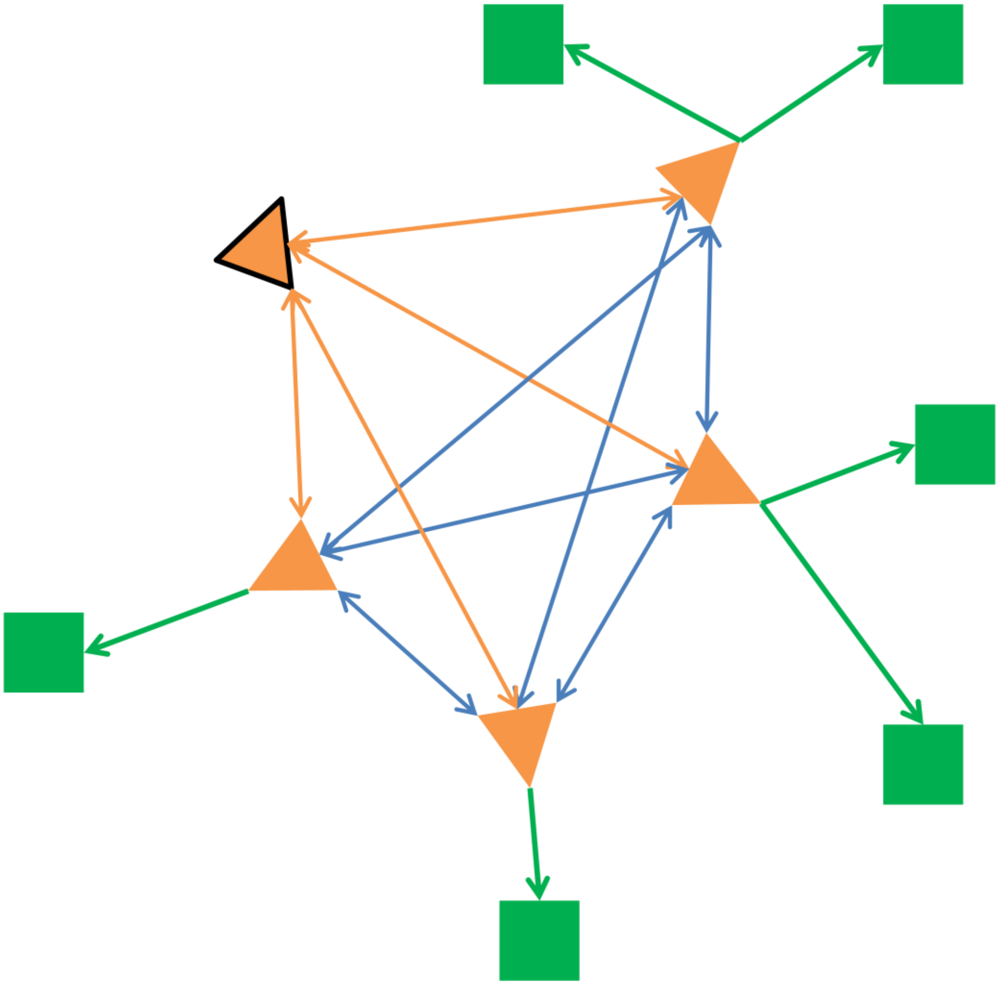


Figure 3 Fully Connected Network allows Immediate Communication of System Changes

This implementation would work well in some situations, but it still has weaknesses. If a vehicle is shot down, it may not be able to communicate its loss to the rest of the team. Until they realized it was gone, its target(s) may be uncovered. Also, the requirement of a fully connected communication network means that no changes can occur in the network. Any communication loss could create a situation in which a change is cannot be communicated rapidly enough to the members of the team. The team members who could change their jamming application to maintain coverage would not know to do this until some target has been ignored for some amount of time.

An example of such a situation is given below in Figure 4. In this case, the previous example has been modified to include an obstruction which alters the communication network. As such the network is no longer fully connected. The same loss of jamming ability for an agent now needs to be communicated over time to the rest of the team. This is represented in Figure 5. Eventually, the loss can trigger a modification in jamming distribution, but this takes some time to occur and in the meantime the target is not being jammed. This means that the mission has failed.

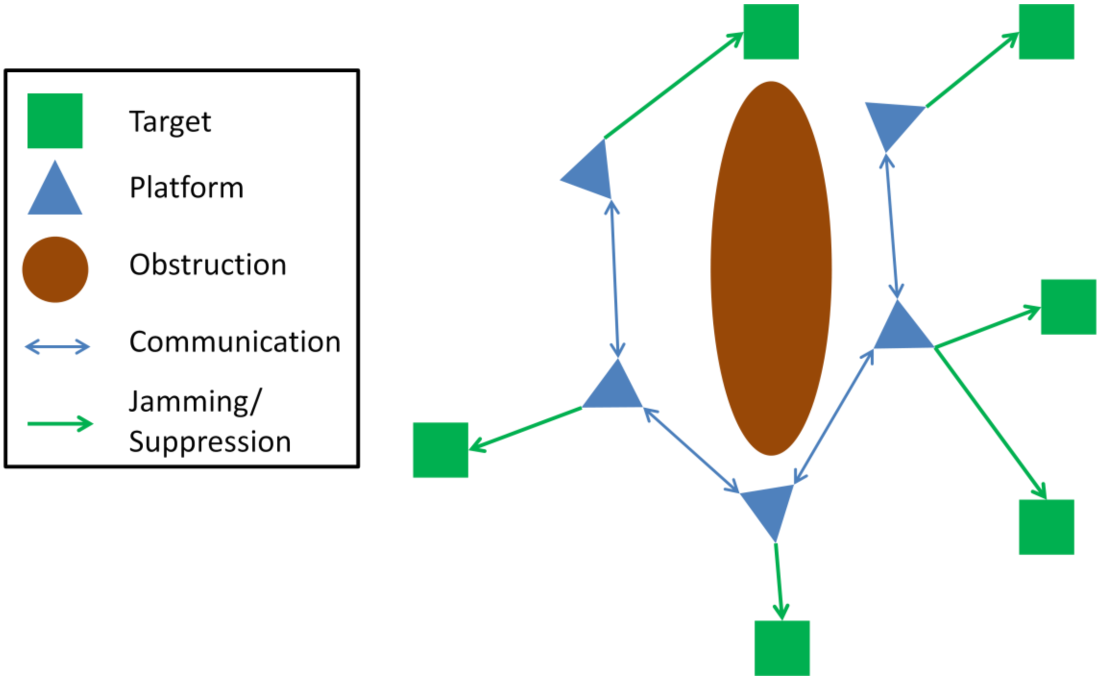


Figure 4 Example Configuration with Obstruction of Communication Network

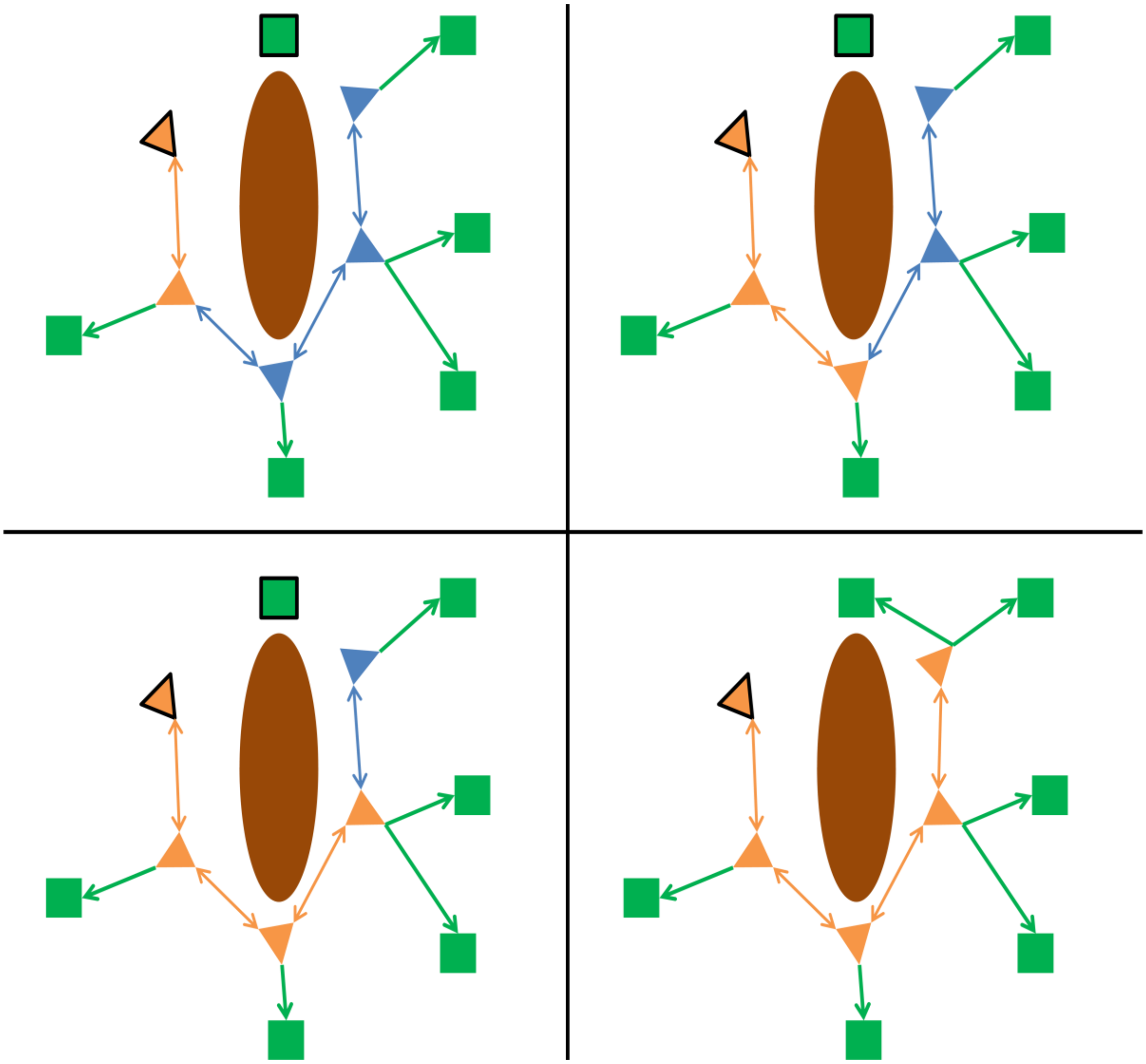


Figure 5 Propagation of Information in Obstructed Communication Network

Non-fully connected networks are handled often in the literature [114] [115] [117] [118] [128] [129] [142] [143] etc, however their use generally requires agreement techniques. These techniques are also ill suited for this type of problem. Consensus building algorithms seek to form agreement through the sharing of information until agents in the team are in agreement to the system information [117] [118] [142] [143] . Such a practice still takes some amount of time and does not ensure that targets will remain covered as the agreement takes place.

The dynamics of the problem are what make it difficult to solve. In this case dynamics does not refer to vehicle dynamics but rather the variability of key problem parameters. This includes the number of UAVs in the system (some may temporarily lose function or be destroyed entirely), enemy targets (which may arise) and losses in communication. These problems are commonly addressed in the literature in one of two ways: a fully connected communication network or implementation of agreement techniques. Both of these options can be successfully used in other problems, but are ill suited for this type of problem. This is mainly due to the strict requirement of constant jamming of all targets. A fully connected network does not allow for network changes to take place, which limits the variability that can take place in the problem. Agreement techniques also have a weakness in that it takes some amount of time for that agreement to occur. In the meantime targets may be ignored, which violates the goals of this problem.

In order to effectively solve this goal, existing techniques must be modified or new techniques developed. Two of the major concerns are better implementing the resource distribution to make the system more resilient to changes that occur, as well as modifying behavior and distribution strategy once a change has occurred.

Such a system would be able to operate under a large amount of changes and disruptions. Any addition or removal of agent might impact performance, but should not cause a failure of the overall system goal. Any addition of tasks into the system should be picked up quickly such that new tasks aren't ignored while the system reacts. Changes or disruptions to the communication network itself may cut off agents from some or all communication from other agents. For this reason old information will be used and must be handled in a way which does not cause system failure. In addition, the rapid addition of new information must be done so properly and effectively. Such information should be used before another change or cut off of communication renders it unusable. The change in performance of the actors as the system operates must also be taken into account and used properly. If one actor's ability changes within the system it might make the current scheme insufficient for completing the overall mission and therefore needs to be changed. Overall many changes can happen in such a system and each of them needs to be dealt with accordingly. Allowing such variability in the system is beyond traditional CC techniques and requires additional methods to ensure both the control and communication operate effectively.

Such a system needs to be very robust and be able to operate under a wide array of situations. The system must have some level of redundancy or ability for agents to pick up the tasks which others are also conducting. This is necessary because any of the changes listed above could make an agent no longer fit to complete its current task. If only one agent is working on a given task, than its removal from this task would cause that task to be unfulfilled. However necessary redundancy is it also comes at a cost, usually of performance. Redundant tasks or operations take up valuable resources which may be better spent doing other activities. The means of understanding what level of redundancy is needed is of importance. When high levels of redundancy are not needed the system should not waste resources in a highly redundant state. However the system should not be so aggressive that it causes failures. Otherwise high levels of redundancy are required for all situations just in case the system might need it. In addition, the system should be able to transition its behavior based on the situation. Despite the fact that the controller has no effect on the changes which may occur within the system, the impact and frequency of such changes do have a dramatic impact on performance. The system must be able to detect when changes occur quickly and transition to a more conservative state in order to maintain mission success during these changes. Conversely, during times in which change does not happen the system should be able to take advantage and increase performance. This should not however make the system more likely to fail if in when changes do occur.

The desire of this research is to pursue cooperative control avenues regardless of any variability which exists and the system. These changes may otherwise cause failures if a traditional CC formulation is used which is why modifications and changes need to be made. Included in this variability are possible changes to the communication network of the system. Despite this included variability of the network it is true that the communication network does have a large impact of the performance of the system.

There is a good reason why much cooperative control research assumes a fully connected network, and that is because of its compactness. A fully connected network means that all possible pair wise nodes are connected, or that communication can be sent immediately from any node to any other without intermediate steps. This has the direct impact that information is as up to date as possible, which as discussed above is a desirable trait to have. Compactness or the ability to receive communication in as few steps as possible is important because it will reduce the amount of old information in the system as well as the age of that information. Systems which are more compact than other then should be able to respond quickly to changes and therefore be able to apply more aggressive. In this way the network can be used as a variable in the control scheme to switch between levels or redundancy to both ensure mission success while increasing mission performance.

1.2 Research Problem

This research seeks to study problem types and aspects not covered by traditional CC. The difficulty of these issues is that it introduces new or more impactful sources of information disagreement which must be properly dealt with to ensure good system performance. Those attributes are discussed below:

- High levels of system variability

    - Changes in the actors of the system, the tasks they need to do, their performance or ability to do these tasks, and how they communicate with each other are all considered. Any of these changes may cause system failures if not properly dealt with.
- Communication
    - Sub-optimal communication directly implies that disagreement of information will exist at some time. Traditional CC schemes use agreement techniques to alleviate this problem. In static cases, it takes some amount of time to achieve an acceptable level of agreement, and it is assumed the time it takes for this to occur does not negatively affect system performance. In more dynamic cases, agreement still takes time to occur, and small levels of disagreement will always exist. In these cases, small levels of disagreement are acceptable.

In order to study and demonstrate the above issues, a proper choice of control problem needs to be made. The purpose of this research is to study the impact of communication on cooperative control and ways of operating in a wide array of variable problem situations. For these reasons the actual control problem of the individual agents as well as the system as a whole should be relatively simple. This will allow for more focused study of communication schemes as well as its impact on the specific control task without that control task taking up the primary research focus.

At this time it is important to discuss the nature of high and low level control. High level control refers to more broad goals of the system, whereas low level control refers to the basic actions which must be taken by an autonomous vehicle. For example, if the goal was to create an autonomous car that can drive from one house to another, a high level controller may give simple waypoints. Turn right on Pine, follow for two miles, turn left on Main etc. A low level controller would be more concerned with the low level activities of the vehicle. Turn the steering wheel 30 degrees for 3 seconds to make a turn, decelerate at x amount etc.

For this problem the main concern is higher level goals such as strategy rather than low level activity such as vehicle movement. This research investigates the impacts of communication issues on the overall system goal which needs to be achieved by a team of agents. This is a high level goal, which can be handled via high level control. The best way to divide system resources is a concern of strategy more so than vehicle dynamics. As such, the high level control will be the focus of this research. This problem is made more abstract by this choice, but it will allow easier implementation of new ideas and quicker simulation times. If these ideas are later implemented in systems with both high and low levels of control, existing techniques such as path planning and target avoidance can be implemented in addition to the strategy based control decisions.

High level controllers are common, especially in autonomous competitions. Primary examples are RoboCup [81] and RoboFlag [115] . These competitions pit teams of autonomous agents vs. each other in a competitive fashion to achieve some kind of goal. The main difficulty of these systems is to determine what strategy works well in a given situation, and when to change strategies. Unlike many low level control problems, the mathematical formalism of high level control is lacking. As such, evaluation of these techniques usually requires simulation and direct experimentation, as opposed to mathematical certainty of performance in low level control. This research is no different, and evaluation of performance must be done via simulation. In order to test possible conditions, a test bed will be created.

The variable nature of the problem to be studied also must have some specific properties in order for it to be amenable to this process. Those properties are as follows:

- Allow for variable amounts of agents which can be added and removed from the system. These agents should be able to immediately perform within the system and be integrated into the team if communication allows.
- Allow for variable amounts of tasks which may be added and removed from the system. New tasks should immediately be picked up and operated on in addition to other pre-existing tasks.
- Be able to facilitate but not require communication. The communication network and schemes are important to the system, but there will be times when it is not possible.
- Agents should have the ability to operate alone and without communication if need be. During times within the simulation it may be possible for specific

agent(s) to be cut off from the rest of the team and should be able perform some part of the mission in this situation.

- Be quick in execution. The control problem must be preformed many times over the course of the simulation to test different problem situations. In addition it must be performed by each agent in the team. A large amount of situations must be studied to test the effectiveness of different cooperative control schemes, and that testing must be done in a reasonable amount of time.
- Be “simple” to implement, as discussed above.

For these reasons the specific control problem chosen is based on assignment optimization. Assignment optimization seeks to match pairs of entities of separate groups in order to maximize or minimize some objective function. For this problem those two groups will be considered agents and tasks. A cost matrix enumerates the ability of a given agent to perform a given task based on a weight between zero and one. A value of zero means that the agent cannot perform that task, while a value of one means that agent is capable of performing the task without any loss of performance. Values in between represent different abilities to perform the given task by different agents; a larger value by one agent represents a better performance of that agent to complete the specific task. The goal of the optimization is to assign each agent to one task such that the overall performance is maximized. Each agent has a limited amount of resources which are used in order to complete these tasks. If an agent does multiple tasks, it must divide this resource amongst them. This indicates the agent completes these tasks at reduced performance as compared to completing a single task. In this case the optimization is linear and can be calculated computationally relatively easily.

Modifications of the basic scheme need to be made in order to more suitably represent this problem. Because of the need for redundancy, multiple assignment needs to be allowed, which means assigning multiple agents to a single task and multiple tasks to a single agent. This also allows for variable agents and tasks. This unfortunately makes a linear problem non-linear, but a simplification can be made at the loss of ensuring optimality of the solution. This is done by decomposing the problem to a series of linear assignment problems and then deciding how to split those resources in the case of multiple assignments. This is an advantageous solution because it allows for the advantages of linearity to remain without much loss. Another source of non-linearity is introduced in order to further punish redundancy and push the system toward the boundary between aggressiveness and redundancy. This is the idea of independent tasking; which means that when multiple agents perform the same task, only the most effective performance of that task counts. Mathematically this is done by taking the maximum of all performances of that task rather than summing them. An example of this is given below in Table 2, Table 3, and Table 4 for a sample cost matrix given by Table 1. Note that the highlighted cells represent the “best” performance of the task or the largest number in the cell.

Table 1 Example Cost Matrix

1.00	1.00
1.00	1.00

Table 2 Example 1 of Task Effectiveness

Assignment		Task Effectiveness	
1.00	0.00	1.00	1.00
0.00	1.00		

Table 3 Example 2 of Task Effectiveness

Assignment		Task Effectiveness	
0.50	0.50	0.50	0.50
0.50	0.50		

Table 4 Example 3 of Task Effectiveness

Assignment		Task Effectiveness	
0.60	0.40	0.60	0.70
0.30	0.70		

It is not necessary to make this assumption, all future results given will still hold true in principle. The values might be different but the idea is the same. This will be discussed below along with the means of determining success of the control scheme within these confines. More details on the specific control scheme will be given in later chapters.

One consideration which must be determined is how often communication occurs with respect to the control action as well as the changes which may occur in the system. For this problem, the most taxing situation must be a possibility which dictates that control action, and changes in the system occur at the same rate as communication. If communication occurs at the same rate as action then the control scheme must act upon old and possible inaccurate information. In order to simplify the problem, each stage of communication and action will occur with certainty as well as the possibility of change in some timescale which will be the basic time step of the problem. The overall problem then will be to chain these time steps together. This is demonstrated below in Figure 6.

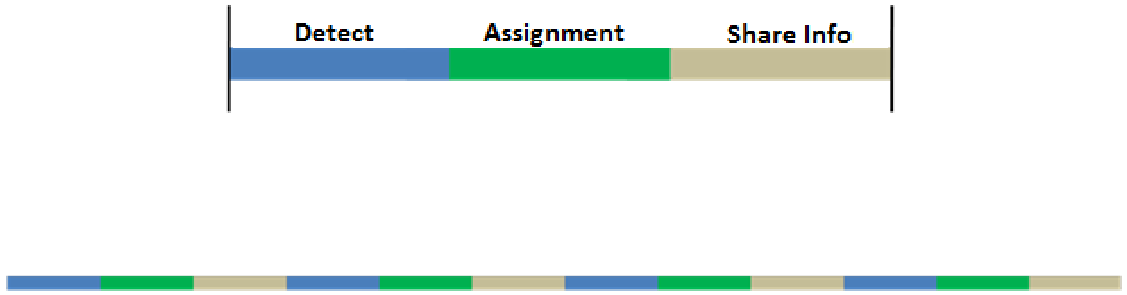


Figure 6 Example of Time-Line and Time Step

The reason why the actions were chosen in this order is because requiring action before communication means that the most immediate changes will only be known the specific agent himself before action needs to be taken. This makes it a harder problem than having the action stage following change and communication which would give the system the ability to more rapidly respond to changes. The more difficult avenue is chosen because the results found for this case should be applicable to less demanding problems, but if a less demanding problem was studied, the converse is not true.

1.3 Problem Considerations in Comparison to Standard Cases

In addition to these choices for the control scheme, a few additional considerations must be made. The complete list of options or choices which must be made is given below with brief description of the impact of each option:

- Communication network
- Variability of system
- Centralized control?
- Frequency of communication
- Frequency of mission action (relative to communication)
- Stringency of agreement

Below in Table 5 are the options for each feature which are captured by existing CC techniques. In this table, green entries mark options easily captured by existing techniques, while yellow options represent ones which may be selected but are limiting. The communication network is usually fixed or allowed to vary in a limited way. The classic fixed network is the fully connected one as discussed above. Non fully connected networks are generally static, but can very under certain conditions. One example of this is the requirement that the network be connected (no agent or team of agents is cut off from the rest). This is important for consensus building techniques which will be discussed in the next chapter. In this way, the system is either not allowed to vary, or may only do so at a low level. Other system parameters are usually kept constant, such as the number of actors in the system.

Table 5 Options Captured by Traditional CC Techniques

Communication Network	Fully Connected	Limited by Control Scheme	Complete Freedom
Variability of System	None	Low	High
Centralized Control?	Centralized	Decentralized	
Frequency of Communication	Continuous	Updated at X Frequency	Only when Changes Occur
Frequency of Mission Action	Slower than Communication	Same Rate as Communication	
Stringency of Agreement	Low	Medium	High

The next consideration is whether or not centralized control can be used. Centralized control is a method of hierarchical control in which one or more agents act as "leaders" of the team. In these cases all of the information is collected by the leaders and distributed to the other members rather than allowing the other members to communicate directly. This can ease the control scheme and agreement, but is more susceptible to failure; if the leader is removed than the system will fail. The other option is to use decentralized control, which does not have the same weakness, but requires a little more effort in the control to ensure effective cooperation and agreement.

The frequency of communication is the next issue. At best for agreement, information will be communicated as often as possible. However, in many more static systems, communication can be viewed as a cost and is not needed that often. In these situations the communication is made at some lower frequency, or only when changes occur. It is assumed that in these later cases the reduction in information will not be a major hindrance to the performance of the control scheme. In the same vein is the frequency of mission action and stringency of agreement. In a way, one can directly impact the other. If information agreement is highly valued, then the system can be made fully connected, or communication can occur so rapidly that any change can be disseminated readily before the information needs to be used. Another possibility is that information may be used before a complete agreement is made, but over time the agreement of information will become better. In such cases, the initial disagreement is not as important and may reduce performance slightly, but not significantly.

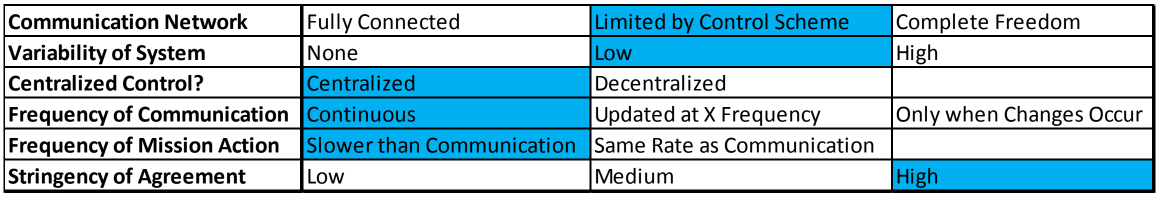
To contrast the desired research objectives vs. what can be done already, the options needed for the purposed control scheme are given below in Table 6. The main difference between the two is the high variability desired in the proposed scheme. This acts as a driver to almost all other options. The specific time-line dictates that mission action occurs at the same rate as communication, which also limits agreement. Agreement will also be made more difficult, as the problem chosen is very sensitive to differences of information. The problems which arise from this disagreement will be a major thrust of the research. Complete network freedom must be allowed, due to this variability. In addition, a centralized control scheme is not possible due to the loss of any agent at any time, including possible leader(s). Finally, while it is possible to have communication occur less frequently, that is not a desirable. As changes may occur at any time, it is desirable to learn about them as fast as possible to reduce the effects of disagreement. On that same note, while it is possible to have communication occur when changes are detected, this is also not desirable. One of the most impactful changes is loss of an agent. In this case, communication would simple cease to come from the lost unit. Without some regular communication from this agent, it is possible that neighbor’s will simple assume nothing has changed. Instead, some frequency of information is needed to know when a change occurs, and that communication should be as frequent as possible for reasons given above.

Table 6 Options Needed to be Captured by Proposed Research

Communication Network	Fully Connected	Limited by Control Scheme	[✓ Complete Freedom]
Variability of System	None	Low	[✓ High]
Centralized Control?	Centralized	[✓ Decentralized]	
Frequency of Communication	[✓ Continuous]	Updated at X Frequency	Only when Changes Occur
Frequency of Mission Action	Slower than Communication	[✓ Same Rate as Communication]	
Stringency of Agreement	Low	Medium	[✓ High]

Of the existing control problems none were found which exactly met the above criteria, however one was found which was closer than the rest. The RoboFlag problem discussed above has many interesting aspects similar to the proposed problem. The system can lose team members (which may be "captured" and removed from the simulation) and has a much more variable communication network than other cases. Perhaps the most important aspect of the problem is a correct determination of strategy and making sure this strategy is followed by as many team members as possible. However, the determination of strategy is conducted via a centralized control scheme. Even though team members can be lost, the system relies on an "outside" presence to determine strategy which may be automated or be a human. Finally communication occurs relatively frequently, so system information may be propagated quickly when allowed by the communication network. These facts are summarized below in Table 7.

Table 7 Breakdown of RoboFlag Problem



Communication Network	Fully Connected	[✓ Limited by Control Scheme]	Complete Freedom
Variability of System	None	[✓ Low]	High
Centralized Control?	[✓ Centralized]	Decentralized	
Frequency of Communication	[✓ Continuous]	Updated at X Frequency	Only when Changes Occur
Frequency of Mission Action	[✓ Slower than Communication]	Same Rate as Communication	
Stringency of Agreement	Low	Medium	[✓ High]

As seen, even the most similar problem still has many differences than the needed options for this research. Also, the basic nature of the control objective is quite different for these two problems; the RoboFlag problem is a competitive competition between autonomous teams while the problem of this research is a resource distribution problem. Some insights in terms of strategy and switching may be gained from the RoboFlag problem, but many of its applications do not extend to this research.

In order to measure the effectiveness of the control scheme a few metrics of interest will be introduced. The most important metric is that of task coverage. It is desired that at every time step, each task has at least one agent assigned to it to complete it with some level of performance. The changes in the system will likely make haphazard control schemes result in certain tasks unassigned, which means that certain parts of the mission are left incomplete. This metric will be the frequency of uncovered or unassigned tasks, which will be given as a ratio of all unassigned tasks to the total number of tasks overall all time steps. Of secondary importance to this is the duration of unassigned tasks. Unassigned tasks are to be eliminated, but if that is not possible then it is desired that they be unassigned for as short a duration as possible.   The other metrics will be used to measure the performance of those tasks, and thus the optimality of the assignment. Those metrics are the average performance level for which all tasks are completed as well as the performance level for the minimum task. This will allow cases in which no failure occurs to be compared and evaluated. Finally, a metric will be used to indicate what percentage of time the performance is equal to the baseline case, what percentage is worse, and what percentage is better. These will be used to give some insights to the system when the above metrics do not tell the entire story effectively.

1.4 Performance Considerations

In order to effectively measure performance other considerations need to be made. The raw values of performance for a given assignment are highly dependent on the number of agents, tasks and the nature of the cost matrix, which may all change within the simulation. Therefore requiring certain performance values of those parameters might be impossible no matter the control scheme implemented. Instead, an idealized case and baseline case will be introduced, from which the actual performance will be normalized. The idealized case will be calculated by using the same techniques used as the actual case, but with perfect knowledge of the system. This will allow the minimum level of redundancy to be used since perfect knowledge allows the system to be the most flexible and respond to any problems. This will represent an unattainable upper bound on system performance. The baseline case will represent one in which communication is ignored and each agent acts on its own without any other information from its team. In this case, each agent must do the task to the best of its ability by assigning itself to every task it can, even at reduced performance by doing so. This case will represent a desired lower bound upon the system. Since the baseline case is easy to implement and requires no teamwork, any performance worse than this will be considered unacceptable. It is important to note that for both the baseline and idealized case there will be exactly zero frequency of unassigned tasks. Any cost matrix for which this is not possible (IE has all zero values for a given task weight) will be considered ill posed. The performance metrics of the actual case will be normalized by the ideal and baseline case as given in below in Equation 1.

Equation 1 System Performance Total Normalization

$$N_T = \frac{A - B}{I - B}$$

This normalization will give a reasonable and understandable metric regardless of the situation and how many agents or tasks there are. All values will be below one, and any non-negative value represents performance at least as good as the baseline case. Any negative values are worse than the baseline and considered undesirable. It is important to note that in some situations of sparse cost matrices that the idealized case and baseline case will be equal. In these situations the performance metrics will be considered 0.5 or the average of the baseline and ideal case. This scheme is called the total normalization because it uses all three communication schemes to create the normalized value.

Another normalization scheme will be used which is presented below in Equation 2. The first normalization gives an idea as to how well the performance is compared to the best case and worst case in terms of communication. This can be somewhat confusing however, which is why a second normalization has been introduced. Instead, a new normalization scheme is devised based solely on the actual and baseline performance. As such it is called the baseline normalization scheme. The baseline normalization takes the actual performance and divides it by the baseline performance at any instant in time. As such, a value greater than one means increased performance, while lower than one indicates lesser performance. In this case, the actual number has clearer meaning as well. For example, a value of 1.2 indicates that the actual case has a twenty percent improvement over the baseline case. This can be thought of in a different way from a design standpoint; what level of agent is needed with cooperation to do the same task as well as an agent without. If lesser agent's performance is increased via cooperation, it becomes possible to create cheaper agents with fewer resources which can do the job just as well as more capable agents without cooperation. For example, if the normalized value is 1.25, that means a newly designed agent with eighty percent of the resource capability can do the same mission at equal performance. Once again, what level of performance increase can be gained from cooperation?

Equation 2 System Performance Baseline Normalization

$$N_B = \frac{A}{B}$$

1.5 Communication Examples and Impacts

In order to effectively summarize and demonstrate the ideas given above, a notional example will be given for the idealized communication case, the baseline communication case and finally a notional normal communication case. Three cases will be given: the initial case will be a standard static condition; the next case will represent a change in the cost matrix and the corresponding responses of each communication case and the final case will represent a loss of an agent from the first case and the corresponding responses. The two normalization schemes can be differentiated easily by their value. For the first scheme, normalized values will never be greater than one, and will most commonly be below .5. For the second scheme, values will likely be larger than one unless the control scheme has poor performance.

For the first static case, the cost matrix given in Table 8 will be used. The cost is designed such that four agents and four tasks exist, and each agent completes each task at levels of poor, adequate, good, and excellent.

Table 8 Cost Matrix A

0.10	0.50	0.70	0.90
0.45	0.65	0.85	0.05
0.75	0.95	0.55	0.15
0.92	0.12	0.52	0.72

For the idealized case, Table 9 represents the assignment matrix for the above cost matrix and Table 10 represents the corresponding task performance matrix. The task performance matrix is created by multiplying the pair wise elements of the cost matrix and the assignment matrix. IE element 1,1 of the task performance matrix is found by multiplying elements 1,1 of the cost matrix and assignment matrix. Of note for the task performance matrix is the highlighted cells in each column. These represent the effective task performance for each task, which is the maximum performance by any agent for that task. Corresponding to the assignment of Table 9 is a graphical example given by Figure 7. In Figure 7 the black filled circles represent agents while white circles outlined in black represent tasks. A link connecting these nodes signifies the assignment between that agent and task.

Table 9 Ideal Assignment for Cost Matrix A

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
0.00	1.00	0.00	0.00
1.00	0.00	0.00	0.00

Table 10 Ideal Raw Task Performance for Cost Matrix A

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.00	0.95	0.00	0.00
0.92	0.00	0.00	0.00

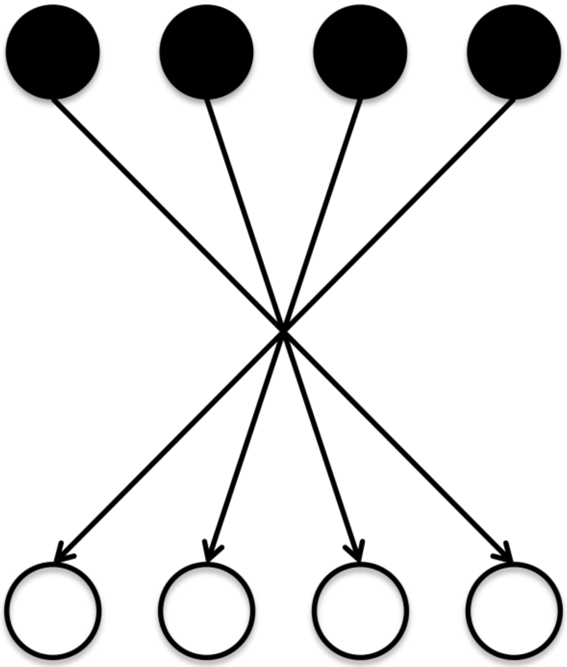


Figure 7 Idealized Graph Assignment for Cost Matrix A

As can be seen, the idealized case completes the assignment with as little redundancy as possible, while still assigning all tasks and leaving none uncovered. The average task performance is .91 with a minimum task performance of .85.

On the other end of the spectrum is the baseline case. The corresponding assignment is below in Table 11 with the raw task performance matrix given below in Table 12. The corresponding graphical assignment of Table 11 is represented by Figure 8. As with the idealized case, the effective task performance is highlighted; as for effective average performance, the other elements will not be counted as discussed above.

Table 11 Baseline Assignment for Cost Matrix A

0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25

Table 12 Baseline Task Performance for Cost Matrix A

0.03	0.13	0.18	**0.23**
0.11	0.16	**0.21**	0.01
0.19	**0.24**	0.14	0.04
0.23	0.03	0.13	0.18

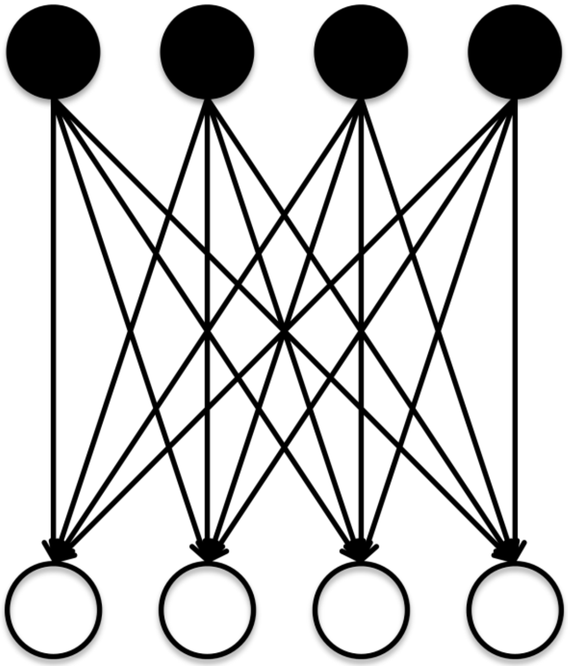


Figure 8 Baseline Graph Assignment for Cost Matrix A

As opposed to the idealized case, the baseline case has the highest level of redundancy. Each agent is assigned to each possible task with reduced effectiveness. This is represented in the assignment by values of .25 instead of 0 or 1 as in the idealized case. Note that the row sum of the assignment must be 1, as this represents the total resource available to each agent.   This case better shows the nature of independent task performance by agents, with non-zero non-highlighted cells effectively being ignored for the purpose of overall task performance. As can be expected by a problem of greater redundancy, the overall performance suffers, with the average task performance of .2275 and minimum task performance of .21.

The normal case without change is given exactly by the idealized case. For these examples it is assumed that the system starts in this state, but changes to the system will demonstrate the difference between idealized and normal communication. The first change will be that of the cost matrix and given by Table 13. The change is relatively small; only one element has changed.

Table 13 Cost Matrix B

0.10	0.50	0.70	0.90
0.45	0.65	0.85	0.05
0.75	0.95	0.55	0.15
0.10	0.12	0.52	0.72

The idealized case assignment is given by Table 14, with the raw task performance given by

Table 15. The corresponding graphical assignment is given by Figure 9.

.

Table 14 Idealized Assignment for Cost Matrix B

0.00	0.00	1.00	0.00

0.00	1.00	0.00	0.00
1.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00

Table 15 Idealized Raw Task Performance for Cost Matrix B

0.00	0.00	0.70	0.00
0.00	0.65	0.00	0.00
0.75	0.00	0.00	0.00
0.00	0.00	0.00	0.72

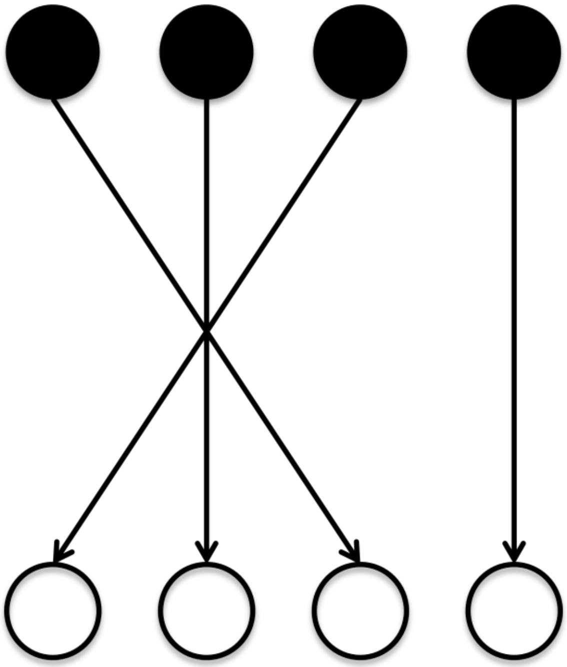


Figure 9 Idealized Graph Assignment for Cost Matrix B

It is important to note in this case that the idealized assignment responds immediately to the changes in the system. This is because it assumes perfect knowledge of the system by all agents so each knows the change happens immediately. This allows it to still complete the goal of no uncovered tasks despite the change. Despite this, performance still drops, with average task completion of .705 and minimum task completion of .65. It is important to note additional meaning to the idealized and actual case. If this problem was to be solved via traditional techniques, one option would be to limit the system to a fully connected network. This would directly correlate to the idealized case. If this is assumed when this assumption is not accurate (the network is not fully connected) and changes do occur, then the system will have problems. This can be viewed in the third case, which dictates normal performance using these ideas.

The baseline case has no real change to its assignment and therefore the assignment matrix and graph are still given by Table 11 and Figure 8 above. What does change is the raw task effectiveness given below in Table 16.

Table 16 Baseline Raw Task Performance for Cost Matrix B

0.03	0.13	0.18	0.23
0.11	0.16	0.21	0.01
0.19	0.24	0.14	0.04
0.03	0.03	0.13	0.18

The average task performance is .2175 with a minimum task performance of .19. Of note in this case is the relatively minimal overall performance between this case and that of before the change. This is due to the increased redundancy of the system compared to the idealized case. It is interesting to note that each case has desirable attributes, the increased performance in the idealized case, and the resiliency of the baseline case.

Next, a notional control scheme will be demonstrated. It will attempt to perform assignment like the idealized case, but with a non idealized communication scheme. This will seek to have minimum redundancy. The actual communication network is given below in Figure 10.

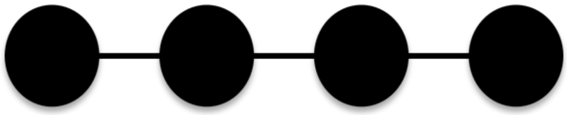


Figure 10 Notional Communication Network for Example Problem

Due to this communication, some agents will be operating on a different cost matrix from its team until the changed information has propagated over the entire system. Before the change occurs the assignment will be exactly that of the idealized case for the first cost given in Table 9, Table 10 and Figure 7. Below is the initial assignment of this system and its task effectiveness given in Table 17, Table 18 and Table 19. Of note is that not all tasks have been assigned an agent, and in Table 18 this task is highlighted orange to reflect this fact. In the corresponding graphical example found in Figure 11 the uncovered task is filled with orange to highlight it. In addition this figure demonstrates which agents are operating on the changed cost and which are operating on the old cost given their color. Grey nodes represent the updated costs and black the old.

Table 17 Normal Case Assignment, First Time Step after Change

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
0.00	1.00	0.00	0.00
0.00	0.00	0.00	1.00

Table 18 Normal Case Raw Task Performance, First Time Step after Change

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.00	0.95	0.00	0.00
0.00	0.00	0.00	0.72

Table 19 Normal Case Normalized Performance, First Time Step after Change

N_T	N_B

Average Task	0.94	3.13
Minimum Task	-0.41	0.00

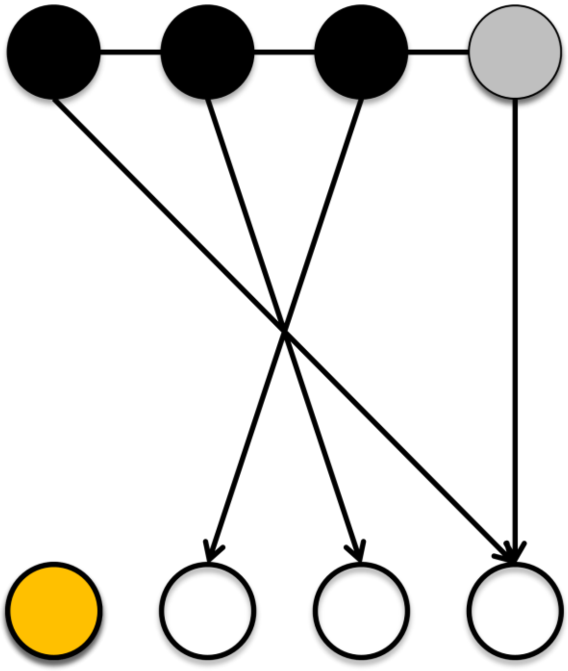


Figure 11 Normal Case Graph Assignment, First Time step After Change

This case shows the first uncovered task, which is precisely what this research is trying to prevent. Essentially what is happening is that the grey nodes are performing the idealized assignment of the new cost, while the black nodes are performing the idealized assignment of the old cost. The problem is that the difference between the two assignments is significant in that each agent will change its task. For this time step, the average task performance is .675 with a minimum performance of zero due to the uncovered task. As the information propagates during the second time step (Table 20, Table 21, Figure 12) and third time step (Table 23, Table 24, Figure 13)

Table 20 Normal Case Assignment, Second Time Step

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
1.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00

Table 21 Normal Case Raw Task Effectiveness, Second Time Step

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.75	0.00	0.00	0.00
0.00	0.00	0.00	0.72

Table 22 Case Normalized Performance, First Time Step

	N_T	N_B
Average Task	0.84	2.90
Minimum Task	-0.41	0.00

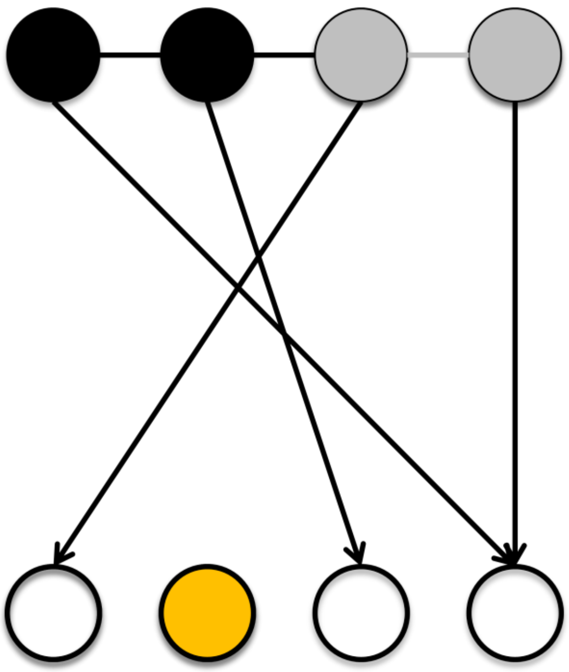


Figure 12 Normal Case Graph Assignment, Second Time Step

Table 23 Normal Case Assignment, Third Time Step

0.00	0.00	0.00	1.00
0.00	1.00	0.00	0.00
1.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00

Table 24 Normal Case Raw Task Performance, Third Time Step

0.00	0.00	0.00	0.90
0.00	0.65	0.00	0.00
0.75	0.00	0.00	0.00
0.00	0.00	0.00	0.72

Table 25 Case Normalized Performance, Third Time Step

	N_T	N_B
Average Task	0.73	2.67
Minimum Task	-0.41	0.00

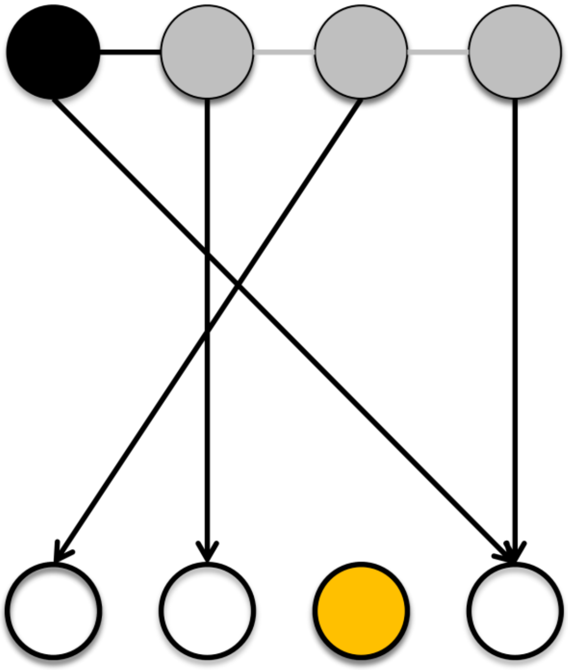


Figure 13 Normal Case Graph Assignment, Third Time Step

For each case there remains an uncovered task, however which task that is shifts for each time step. For the second time step the average task performance is .625, and .575 for the third time step. This example demonstrates the need to have agreement in system data (specifically the cost matrix) for aggressive assignment schemes. In this case only one element changed, and yet it causes such a problem in the system. The final assignment is the same as above in Table 14, but the final graph assignment is given below in Figure 14. This highlights that when the information each agent is acting upon is finally in agreement the control scheme works without uncovered enemies. This inspires the question of how to determine when agreement is in place, and what to do in these situations.

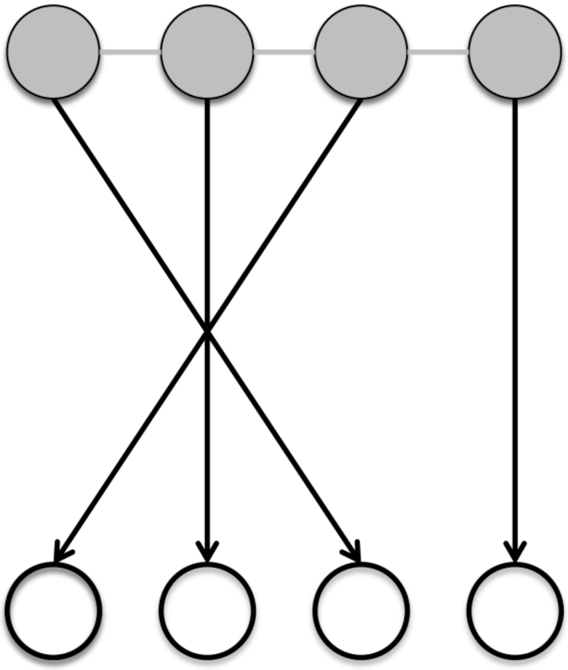


Figure 14 Normal Case Graph Assignment, Final Time Step

Another example of a system change is given below. The initial cost matrix is the same as given above in Table 8. In this case the change represents the loss of an agent, removing of the last row in the cost matrix of Table 26.

Table 26 Cost Matrix C

0.10	0.50	0.70	0.90
0.45	0.65	0.85	0.05
0.75	0.95	0.55	0.15

For the idealized case, the system responds immediately to these changes. The assignment is given in Table 27 with the raw task performance given in Table 28.

Table 27 Idealized Assignment for Cost Matrix C

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
0.50	0.50	0.00	0.00

Table 28 Idealized Raw Task Performance for Cost Matrix C

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.38	0.48	0.00	0.00

For this case it is important to note that the difference in number of agents and tasks requires one agent to perform multiple tasks or else have a task uncovered. Due to the independence of tasks preformed by the different agents it is better to have only one agent perform multiple tasks rather than all agents perform multiple tasks. The average task performance is .6525 with a minimum performance of .38. The graphical assignment is given below in Figure 15. One other thing to note from this case is the dramatic changes in task performance that come from these changes, even for the idealized case. This reinforces the idea of metric normalization which will to measure control effectiveness later in the research.

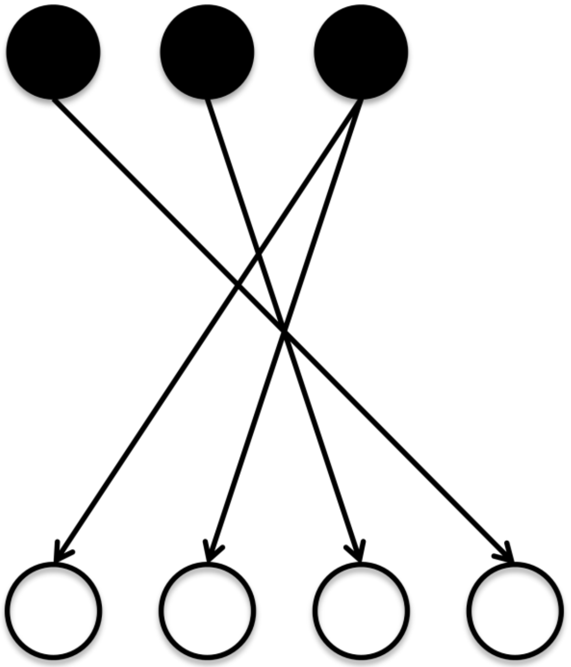


Figure 15 Idealized Graph Assignment for Cost Matrix C

Once again the baseline case has some decreased performance but also great resilience to changes. The assignment is given in Table 29, the raw task performance in Table 30 and the graph assignment Figure 16. The average task performance is .2175 with a minimum task performance of .19.

Table 29 Baseline Assignment for Cost Matrix C

0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25
0.25	0.25	0.25	0.25

Table 30 Baseline Raw Task Performance for Cost Matrix C

0.03	0.13	0.18	0.23
0.11	0.16	0.21	0.01
0.19	0.24	0.14	0.04

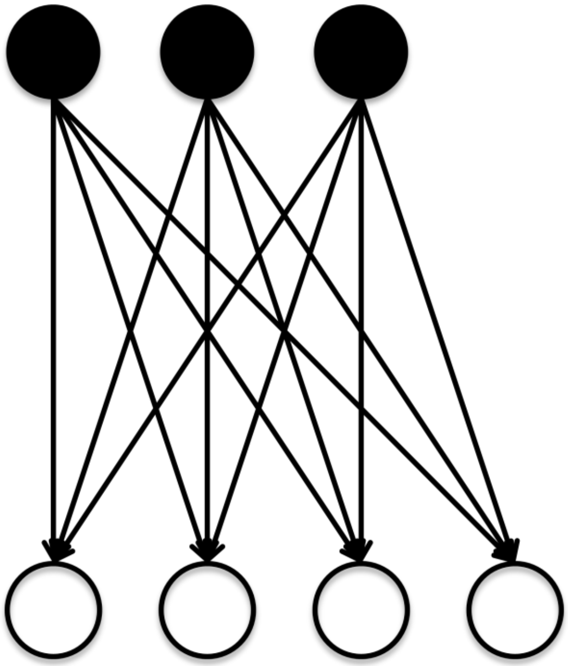


Figure 16 Baseline Graph Assignment for Cost Matrix C

For the normal case problems arise again. The assignment is given by

Table 31 with the raw task performance given by Table 32.

Table 31 Normal Case Assignment Immediately After Change

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
0.00	1.00	0.00	0.00

Table 32 Normal Case Raw Task Performance Immediately After Change

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.00	0.95	0.00	0.00

Once again there is an uncovered task. The average task performance is .675 with a minimum task performance of zero due to the uncovered task. The graph task assignment is given below by Figure 17. The system can respond after only one more time step and will have performance equal to the idealized case. This example differs from the above case because it recovers after only one time step, and without full agreement of information. This is demonstrated by Figure 18, where only one of the remaining agents has the accurate knowledge of the system but still can complete the overall goal. All time steps after this (assuming no other changes occur) will not change the assignment or performance. This case demonstrates a different problem with the system than the case above. Even though there is not information agreement this is not really the main problem, but rather an inability to respond quickly to changes. This is directly tied to the assumption made earlier about the sequence of events within a time step. The more restrictive option was chosen, which essentially means that any change cannot be communicated to other agents in the team until after an assignment has taken place and therefore at least one time step until any change is known. This is especially relevant when an agent is lost; in cases where it is not lost that agent itself will have knowledge of the change and be able to respond. When the agent is lost no other part of the system has knowledge of this until no communication is received from this agent. In this specific case, has the less restrictive assumption been made, the system would skip the configuration of Figure 17 and go directly to Figure 18 due to that communication happening before assignment is required. This demonstrates the concept of more limiting assumptions and how the system is affected by them.

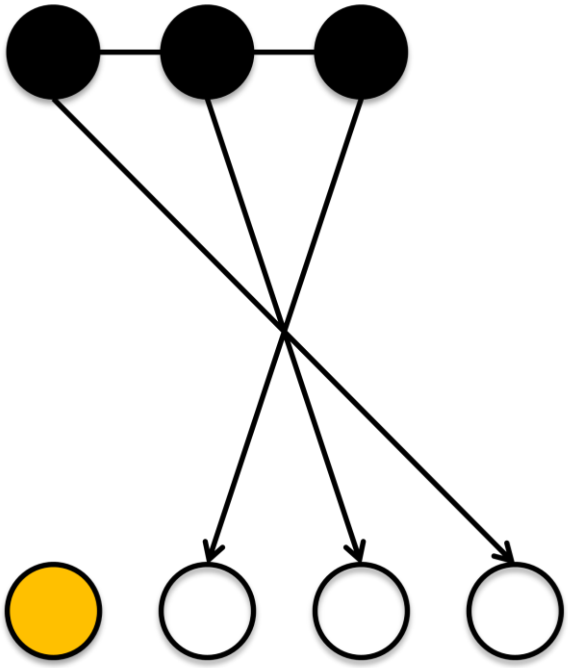


Figure 17 Normal Case Graph Assignment Immediately After Change

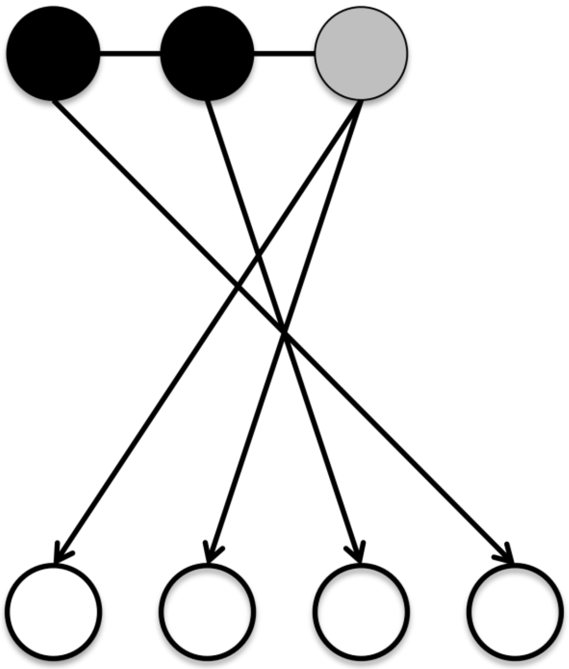


Figure 18 Normal Case Graph Assignment, One Time Step after Change

These examples show the need for a more sophisticated control scheme in order to adapt to the issues of loss and change within the system. Two of the most impactful problems in this system are the need to operate without system agreement and a reduced ability to immediately respond to all system changes.

1.6 Assumptions

While it is desired that the system free of assumptions, some must be made to scope the problem and make sure that it is well posed. Analyses relaxing some of these constraints will be conducted, but not to the same extent as the main thrust of the research.

- Each task must be able to be assigned to at least one agent
    - Otherwise all cases including baseline will fail; if no agent can be assigned to a given task, it will always remain unassigned.
- Changes may be as dramatic as going from the maximum to minimum number of agents or minimum to maximum number of tasks in one step
- Communication: No errors beyond those given already in the system (which may simulate false positives, losing communication etc)

1.7 Aspects of the Problem

Another summary of the problem can be given by its important aspects which are summarized below:

- Combinatorial

- Limited resources
- Uncertainty
- Information gaps
- Incompleteness in information
- Disruption
- Failures
- Scaling implications

1.8 Other Similar Problems

While the system is similar to many problems which will be discussed in the background research section, it also shares some similarities to common problems which will not otherwise be mentioned in this document. Those are as follows:

- Byzantine Generals Problem
  - This problem deals with communication and agreement over a networked system. The problem can briefly be summarized as follows: A general wants to give orders to his troops through his officers, however he knows there are those in his ranks who would try to subvert his wishes and give false orders to those below him. The problem is how to develop a means

    - to ensure the proper orders are delivered and not false ones? Generally this problem is solved via voting and requirements of unanimous agreement at various levels. This is interesting because it gives an alternate means of agreement in a system without it, but this problem assumes that the agents of the system do their best to communicate accurate data and would not subvert the desired goal. Voting over old information does not help in this case, as new information is always as accurate as possible.

- Internet Routing Protocols/Ad hoc wireless networks
    - These problems deal with changing communication networks and trying to quickly and efficiently determine the new network and not waste redundant communication. These networks have nodes which often are added or removed to the system and therefore knowing the network and therefore being able to find the most effective path between any two nodes is important. Wasted communication is defined as that not necessary to transfer the desired communication and is reduced or eliminated if possible. The problem of this research is similar in that the changing network and trying to learn that network. However in this research the overall goal is control across the network rather than just communication. This adds the requirement of more consistent communication for that control, even though it is redundant otherwise.

This information leads to the overall focus of this research and the questions it will study and answer.

1.9 Research Questions

1. Can a metric or set of metrics be found which accurately represent the dynamic effects and changes of the system as it pertains to Cooperative Control? Can these metric(s) be used as a means to trigger changes in the control scheme?
2. Can a control scheme be developed which will consistently perform no worse than the baseline scheme of no communication?

1.10 Hypotheses

1. Yes. It is believed that existing metrics such as diameter as well as new metrics based on the rate of change of the system can be used to describe the system and therefore a control system can be based on them.

2. Yes. By using shared information, it should be possible to outperform situations in which information is not shared. In some situations the sharing of information may not be able to keep up with the changing system, but special considerations can be made during those cases to improve performance.

1.11 Outline of Dissertation

This document will be broken up into six chapters in addition to this one. Each chapter covers a separate are of the research and each will be briefly described below.

Chapter two is a discussion of the background information which will serve as the inspiration and foundation of this research. It is broken up into three major areas: network analysis and theory; cooperative control and assignment optimization. These three areas make up the control problem, the actual cooperative control of the team of agents and the foundation from which the control will be based and adapt. Each section will go over mainly what exists from each field, and later is applicability.

Chapter three discusses the overview of the test bed from which the developed cooperative control methods will be tested and evaluated. This environment will give a means to test a wide variety of situations in terms of network types, impact of changes, frequency of changes and numbers of actors. This chapter discusses the issues of the selected research problem and how those are account for in the test bed. Final a discussion of visualization techniques to aid in understanding of the problem is presented.

Chapter four discusses the various issues of the cooperative control technique with specific focus on agreement between the agents of the control team. Agreement is perhaps the most important problem and requires special consideration within such a variable problem. In addition, the method of control will be introduced and discussed. The control will still need a mapping from the network variables to those needed for control, and that requires special testing which will be the focus of the next chapter. The determination of the important network metrics will be determined in this chapter.

Chapter five is focused on the tuning of the control mapping based on key network metrics. In order to make this process easier, the test bed is modified to induce the largest amount of changes for failure to occur. Failure may only occur during certain parts of the communication and control process. By choosing a specific frequency of changes these phases occur in higher percentage than otherwise. In addition, a certain major change in the system will be discussed, and it will be given special consideration in terms of how it affects the cooperative control process.

Chapter six discusses the results of the overall cooperative control scheme devolved. The scheme and performance cannot be fully explored by the standard test however, and must be explored by additional problems which are discussed.

Chapter seven focuses on analyzing weaknesses in the control scheme presented by relaxing some of the assumptions made in the system. Using the standard assumptions of the problem, the devised technique is complete. However, by expanding use to other conditions some weaknesses are found. Certain assumptions and modes of the problem are changed or relaxed to introduce more system problems of importance. These new problems require modification to the control scheme and their implementation is discussed. Finally the results and key factors of these new problems are given.

Chapter eight is the conclusion of this research. It will go over the key areas of interest, highlight important results and discuss the most important contributions of this research. Finally, areas of future work are discussed.

CHAPTER 2: BACKGROUND LITERATURE REVIEW

This chapter discusses the background and previous studies which feed into this research. The background will focus mainly on three areas: network analysis and graph theory; cooperative control and assignment optimization. The first two areas will lay the foundation for the cooperative control scheme which will be based on the state of the communication network. The final area will give the foundation for the specific control problem itself.

2.1 Networks and Graph Theory

The study of networks is a fairly recent one, mainly coming about in the last century, with a strong rise in research in the last ten to twenty years due to its profound impact on a number of varied problem areas. Mathematically, the study of networks is part of a larger field called Graph Theory and can be traced back to the 18th century with work from Leonid Euler. To avoid some confusion, the term network and graph are somewhat interchangeable based on the application. While the term graph is mostly used to describe the mathematical concept and network is used to describe a physical system, these are not hard set rules. Early network analysis work done by Erdos and Renyi in the late 1950s [45] and this analysis studied certain types of graphs that were formed by various probabilistic rules called random graphs. Erdos and Renyi found that these graphs had certain interesting results based on the probability of connection between entities within the graph and the number of entities itself. They found that for certain thresholds, the final graph would have drastically different properties, based on the connectedness between those elements. The next important work on Networks was based on a phenomenon found in nature rather than mathematical proofs, one which we now call “6 Degrees of Separation” or the “small-world effect”. In the late 1960s, Stanley Milgram [97] , [136] devised an experiment in which he sent 160 letters to people in Omaha Nebraska with the final goal of those letters reaching a specific person in Boston Massachusetts. The instructions told them to send those letters to an acquaintance they knew who would be more likely to move the letter to its desired target. That person would then do the same, until the letter finally reached the person of interest. While this experiment has been criticized for various reasons, the results were astonishing. Milgram found that on average it only took six sendings before the letter made it to the person of interest, which gave rise to the concept of “Small-World” and “Six degrees of separation”, which is to say that despite the large number of people in a society, the actual distance between people via their relationships is much smaller than the size of the population. During this time the study of networks was either a social or purely mathematical one, but the two sides did not connect until the late 1990s, when Barabasi, Watts, Strogatz and other researchers began to look at the scaling of networks and real life networks (such as the internet). From this research came some dramatic results, perhaps the most famous and celebrated being that of a class of networks called scale-free networks. A class of networks was found to have a “scale-free” property because some properties did not changed as the size of the graph changed. Specifically, scale-free applies to a family of graphs whose degree distribution follows a power law, IE $P(k) = ck^{-\gamma}$. These graphs are mainly dominated by a small number of highly connected elements, and are commonly found in social systems. Work was done to determine how networks of this type are created in nature, and devised some growth models based on preferential attachment. This means that new nodes introduced into the network are more likely to link with nodes with large amounts of links. This vaulted network research and can be considered the beginning of modern network analysis.

Many good survey papers for graphs and networks exist with different foci. One of the best reviews in terms of describing the issues in a clear way for someone outside of the field is given by Newman [104] , and he discusses what is important, what is missing and some critical applications. Boccaletti et al. [13] gives a very good general overview mainly discussing features and elements of graphs. Albert and Barabasi [2] mainly discuss the structure of graphs and how certain networks grow, mostly discussing the idea of preferential attachment. Dogorotsev and Mendes [42] give a good background of evolutionary trends in graphs, how growing graphs in certain ways can provide meaningful results. It is by no means a survey of the field, but a good, short and simple description of networks and why they are important is given by Strogatz [134] .

2.1.1 Mathematical Definition of Graphs

Mathematically speaking, a graph is defined as a collection of nodes, or vertices which can be connected via links or edges. Figure 19 is an example of a simple graph, where nodes A, B, C, D are connected via various links. In case R node C is disconnected from the other nodes.

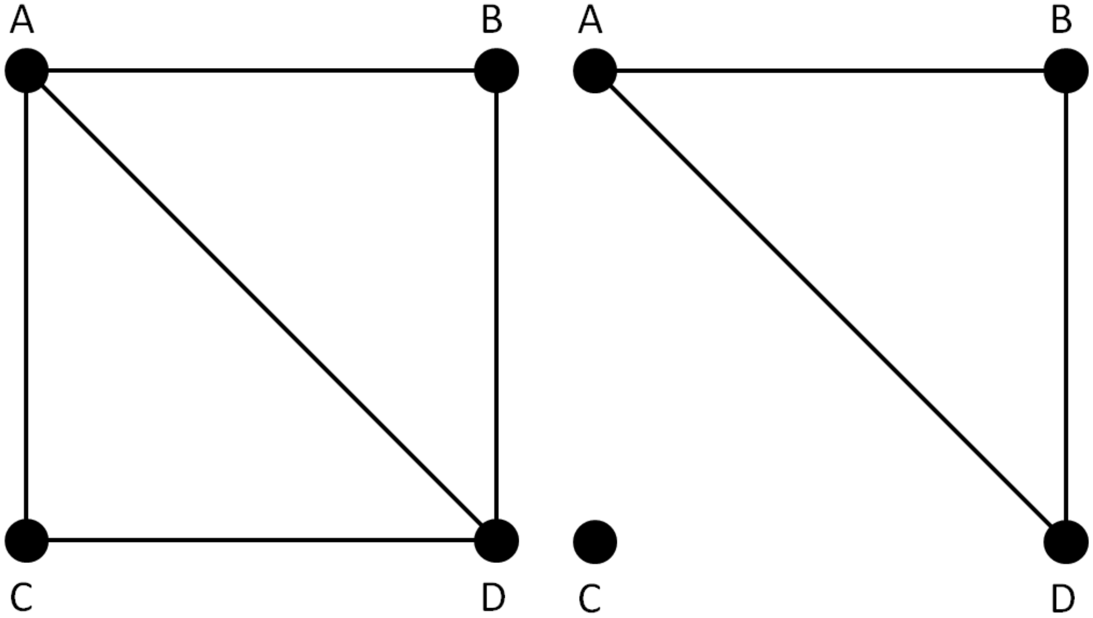


Figure 19 Example Graphs left (L) and right (R)

Graphs are traditionally defined in one of two ways, either via a paired set or by an adjacency matrix. For Graph L of Figure 19, that representation would be [{A,B}, {B,D}, {C,D}, {A,D}, {A,C}] while in Graph R the representation is [{A,B}, {B,D}, {A,D}, {C}]. In this case the order of the pair does not matter. The adjacency matrix representation for both graphs can be seen below in Table 33. In this case, if node pair i,j is linked, matrix element i,j has a 1, otherwise the value is 0.

Table 33 Adjacency Matrix for Graphs left (L) and right (R)

0	1	1	1
1	0	0	1
1	0	0	1
1	1	1	0

0	1	0	1
1	0	0	1
0	0	0	0
1	1	0	0

Another type of matrix used for the study of graphs, but more so for analysis rather than representation is the Laplacian matrix. The Laplacian matrix is the same as the adjacency matrix, except the diagonal elements are the negative of the number of links connected to that node (IE the rows should sum to zero). The Laplacian matrices for graphs L and R are given below in Table 34.

Table 34 Laplacian Matrix for Graphs left (L) and right (R)

-3	1	1	1
1	-2	0	1
1	0	-2	1
1	1	1	-3

-2	1	0	1
1	-2	0	1
0	0	0	0
1	1	0	-2

In these cases, the links are un-weighted and undirected; in general this may not be true. Graphs with weighted links are called weighted graphs, and graphs with directed links are called directed graphs. Graphs with both directed and undirected links are called mixed graphs. Also, there is a maximum of one link per any given pair of nodes, and no nodes have a link which connected at both ends to itself. Such graphs with these four conditions are called simple graphs. When graphs have multiple links between a pair of nodes, they are called multigraphs. When conditions such as these are present in a graph, the definition must change to reflect this. This is generally done within the adjacency matrix, where non unity values are used for weights, and a directed graph will no longer have a symmetric adjacency matrix or Laplacian matrix. For this research, all graphs are simple unless otherwise stated. Some other examples of special types of graphs are fully connected graphs and bipartite graphs. A fully connected graph is one in which each node pair is connected by a link, or likewise that every possible link in the network exists. These types of graphs are commonly found in communication networks where there is full communication between vehicles. An example of a fully connected graph and its adjacency matrix is given below in Figure 20 and Table 35.

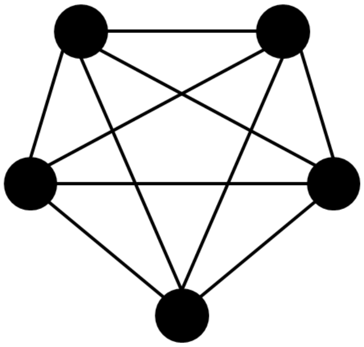


Figure 20 Example Fully Connected Graph

Table 35 Adjacency Matrix of Example Fully Connected Graph

0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

A bipartite graph is one in which the vertices may be divided into two disjoint sets. Each set of vertices may only be connected with those in the other set and not within its own set. One key feature of a bipartite graph due to its structure is that no odd length cycles exist which essentially states that all graphs of this type have a connectivity of zero both locally and for the entire graph itself. If the two sets of vertices are of equal number, the graph is called a balanced bipartite graph. An example of a bipartite graph and its adjacency matrix is given below in Figure 21 and Table 36. Notice that the structure if the adjacency matrix. For a bipartite graph having sets U and V, of size n and m, the upper left n x n block and the lower right m x m block will be all zeros. Connections will only appear in the upper right n x m block and the lower left m x n block.

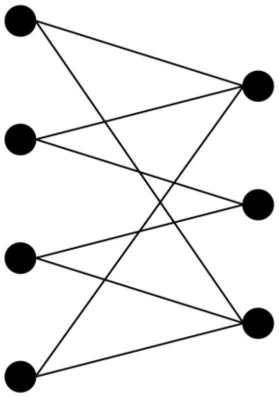


Figure 21 Example Bipartite Graph

Table 36 Adjacency Matrix for Example Bipartite Graph

0	0	0	0	1	0	1
0	0	0	0	1	1	0
0	0	0	0	0	1	1
0	0	0	0	1	0	1
1	1	0	1	0	0	0
0	1	1	0	0	0	0
1	0	1	1	0	0	0

2.1.2 Measures

Many techniques exist to analyze graphs, mainly separated into important metrics or measures and analysis of matrix representations not the least of which is spectral graph theory. First, a list of important measures is given. This is by no means a complete list; new measures are created for different applications. For all cases, a simple graph is assumed, but many metrics have extensions for weighted and/or directed graphs. Please see Balakrishnan and Ranganathan [7] and Berge [12] for more background and detailed information on graphs. Below are example graphs of a ring and spoke network which will be used to illustrate some of the measures more clearly (in Figure 22 and Figure 23). These graphs are finite for visual reasons, but can be extended to have N number of nodes (for the spoke network extension there remains only one node that is connected to the rest).

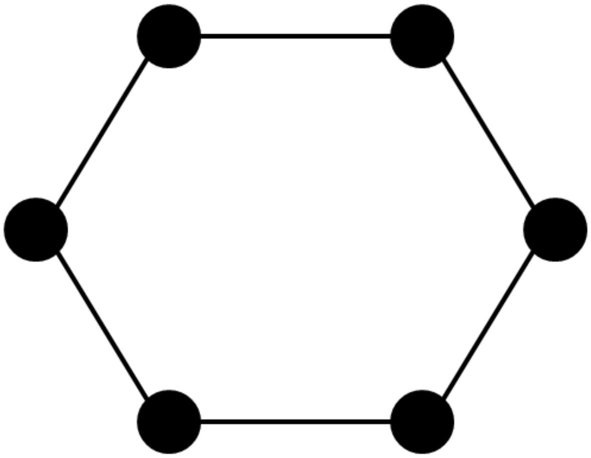


Figure 22 Ring Graph

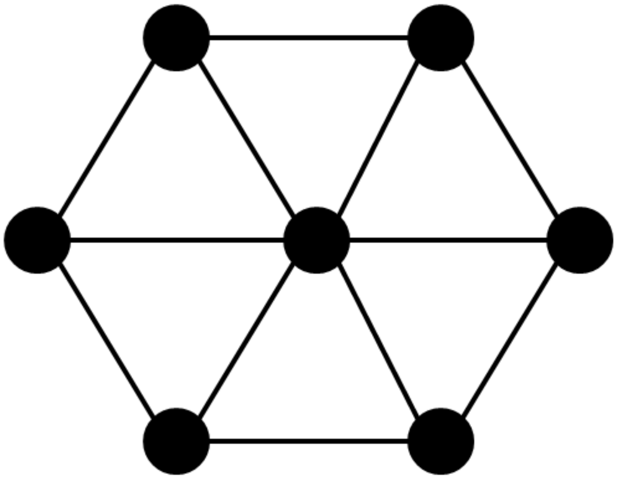


Figure 23 Spoke Graph

Degree (Degree Distribution) – The degree of a node is simply the number of links connected to that node. The degree distribution is the distribution of all node degrees. The degree distribution of the ring graph is [2, 2, 2, 2, 2, 2] because all of the nodes have degree two. If the graph is extended to a larger number of nodes, the degree distribution would be the same only with more elements. The degree distribution of the spoke graph is [3, 3, 3, 3, 3, 3, 6]. If this graph were to be extended, the distribution would be [3, 3, …, 3,N-1] where N is the total number of nodes in the graph. The degree distribution is commonly used and cited as one of the main descriptors of a graph, but recent research suggests that there can be great differences among graphs with equal degree distributions.

Path Length – The distance between nodes (if a path exists) as measured by the number of links required to traverse the path. The shortest path length is often of key interest if multiple paths exist. The path lengths range from one to three in the ring network (or one to N/2 in the N dimensional network), and from one to two for a spoke network of any size.

Connectedness – Two nodes are considered connected if there exists a path between those two nodes. For a graph to be connected, all node pairs must be connected. For a non connected graph, there exist connected components, in which a subset of nodes is connected. In general the maximal or unique component is often of interest. For directed graphs, connectivity can be further divided into strongly and weakly connected. Strong connectivity implies that for each node pair u and v, a path exist from node u to node v as well as from node v to node u. Weak connectivity means the graph does not have the property of strong connectivity, but if the directed links are replaced by undirected ones, that graph would now be connected. Each of the example graphs are connected, while the right graph of Figure 19 is disconnected. Within the right graph of Figure 19 nodes A, B, D form a large component of the graph since it is larger than the average number of nodes and a graph formed of only these nodes is connected.

Diameter – The maximum shortest path length among all node pairs. Only applicable if the network is connected (IE the diameter is finite and defined). The diameter of the ring network is three or N/2 and is two for the spoke graph of any size.

Radius – Involves the idea of a “central” node or set of nodes. To determine the radius of a graph, for each node, create a list of the minimum path lengths for all other nodes and determine the maximum length and associating it with that node. The minimum value of that measure among all nodes is the radius. In simpler terms, it is the node which is “closest” to all other nodes. For the ring network, the radius is equal to the diameter, and for the spoke network, the radius is one (always for the central node).

Clustering Coefficient – Exists for both individual nodes and the global graph. Nodal connectedness represents how close a node’s neighbors are to being a complete component. IE if all of a node’s neighbors are neighbors of each other, than that node’s clustering coefficient is one. For any node of the ring network, the local clustering coefficient is zero since its two neighbors are not connected. For the spoke network, each outer node has a local connectivity of two thirds, while the inner node has a local connectivity of two fifths (or 2/(N-2)). Global connectedness can either be thought of as the average of each nodes nodal connectedness or the ratio of the number of closed triplets (3 nodes all linked to each other) to the number of connected triplets (3 nodes connected to each other). In general these two measures are not equal so it depends on the application as to which is used, but unless otherwise stated, global connectivity will mean the latter definition in this research. For the ring network, the global connectivity is zero. For the spoke network, the global connectivity is 6 (N-1).

Cut Vertex (Set) – A cut vertex is a vertex whose removal increases the number of connected components in a graph by disconnecting sections of that graph. The links connected to the vertex of interest are also removed with the vertex itself. If the graph is connected removal of this node makes the graph unconnected. Sometimes a cut vertex set is of interest, which is the minimal set of vertices needed to be removed to make the graph disconnected. For the ring network, any non neighboring vertices are a cut vertex set, and for the spoke network any two non neighboring outer nodes in addition to the inner node form a cut vertex set.

Cut Edge (Set) – A cut edge is similar to a cut vertex, but for an edge instead. A cut edge is an edge whose removal increases the number of connected components in a graph by disconnecting sections of that graph. Sometimes a cut edge set is of interest, which is the minimal set of vertices needed to be removed to make the graph disconnected. For the ring network, any two edges form a cut vertex set, and for the spoke network to disconnect N outer nodes from the graph (but not from each other), the minimum cut edge set number is N+2 (N to remove all nodes from the central node, and two to remove the additional edges connecting to the rest of the outer nodes)

Cheeger Constant – The Cheeger constant pertains to the same idea of a cut edge set, but is a more elegant mathematical definition and has other applications in graph theory and Riemann geometry [24] . Mathematically, it can be defined by Equation 3 below:

Equation 3 Cheeger Constant

$$h(g) = min\left(\frac{|\partial A|}{|A|}, 0 < |A| \leq \frac{|V(g)|}{2}\right)$$

Where A is a subset of vertices in the graph (the total set is V(g)), and ∂A is the set of links which have one end within the set of vertices A, but the other end is not in A (IE links that connect A to the other portion of V(g)). In other words, it is the largest component that is less than or equal to half the total graph which is the least connected to the other part of the graph. Values are above zero for all connected graphs, and smaller values represent bottlenecks or small cut edge sets, where large values mean the various parts of the graph are well connected to each other and overall the graph is has high connectivity. For the ring graph, the Cheeger constant is two thirds (2/N), and for the spoke graph the Cheeger constant is five thirds ((N/2+2)/(N/2)). This demonstrates the importance of graph structure when growing graphs.

S-Metric – The "structural" metric was created by Li [92] as a way to find a better metric to describe network structure beyond the commonly and almost ubiquitously used degree distribution.   Li found that by using a common degree distribution and varying other parameters, that drastically different networks could be formed. An example of different networks sharing a common degree distribution is given below in Figure 24.

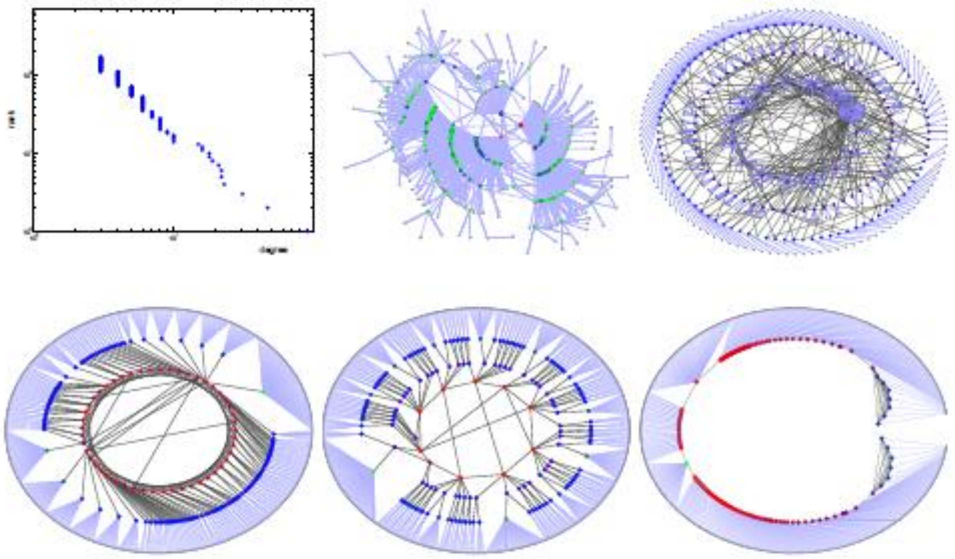


Figure 24 Comparison of Vastly Different Graphs with the Same Degree Distribution [92]

As an attempt to solve this problem, a new metric called the structural (s) metric was developed by Li [92] . This metric is given below in Equation 4,

Equation 4 Structural (s) Metric

$$s(g) = \sum_{i\epsilon V}\sum_{j\epsilon V}\frac{1}{2}d_i a_{ij}\, d_j$$

Where d_i is the degree of node i and a_{ij} is the component i,j of the corresponding adjacency matrix. In layman's terms, the structural metric is a measure of how connected the largely connected nodes are to each other. For scale free networks, the s-metric has connections to other various notions, such as self-similarity, likelihood and assortivity.

2.1.3 Analysis Techniques

Spectral graph theory can be simply summarized as the study of the eigenvalues and eigenvectors of matrix graph representations and how they pertain to behavior of the graph itself. The two most commonly used matrices for analysis are the adjacency and Laplacian matrix described above. A key interest of this spectral graph theory is how the eigenvalues are bounded, for details of this please see [34] . One of the important keys of spectral graph theory is that the choice of labeling of the nodes in a graph will change both of these matrices, but the spectral properties of both are invariant to this choice. A few key properties of both matrices will be discussed.

The adjacency matrix has some interesting properties, mostly for simple graphs. Due to the nature of the adjacency matrix for an undirected graph, the adjacency matrix will be symmetric with all real valued entries, meaning that the equivalent eigenvalues will all be real and the eigenvectors will form an orthogonal set. Another interesting property involves the multiplication of the adjacency matrix with itself. If A is the adjacency matrix, then for A^n, element i,j represents the number of path lengths from node i to node j of length n. This is useful for determining the number of triangles within a graph (used for calculating global connectivity) by taking the trace of A^3 and dividing it by 6 to eliminate extra counting. In addition, the energy of a graph is taken to be the sum of the eigenvalues of the adjacency matrix.

The Laplacian matrix is more commonly used as compared to the adjacency matrix. Similarly to the adjacency matrix, it is also symmetric for undirected graphs, and for such cases it is always positive semi-definite. The Laplacian matrix will always have at least one zero eigenvalue, due to the nature of the matrix, because the [1,1,…,1] vector corresponds to the null space of the matrix. In fact, the number of zero valued eigenvalues is a direct measure of the number of connected components within the graph. If the eigenvalues are ordered from smallest to largest, the second smallest eigenvalue corresponds to the algebraic connectivity of the graph, which is another measure of connectivity. It is bounded above by the traditional global connectivity, and is often used to study the synchronization of a graph.

Thus far all description and analysis of graphs has been for static systems. Dynamic graphs are becoming more popular research areas [127] , [96] but the amount of supporting work is much smaller than that for static graphs. Issues such as stability about an equilibrium of the graph are of main concern, but full extensions of graphical stability to Lyupanov stability have been made. Among analysis of dynamic graphs is work by Siljak [127] and Zecevic [146] . In these works, dynamic graphs are looked at for various reasons from stability to modeling Boolean networks. For the stability section, a measure is created to determine the difference between two graphs and is used to measure equilibrium and overall stability in the traditional sense. One limitation of this is the assumption that the change in the network is a continuous function of time and the current state of the network (in essence an abstracted ordinary differential equation). This same assumption is also made during the Boolean network modeling. In addition, in the more general sense of stability, the choices of measure to use are left to the individual.

2.1.4 Structure and Growth of Networks

One of the major research areas among networks deals with structure. What are the key elements of a network, what are commonly found substructures and other such questions are key in research. Watts and Strogatz [141] discussed various types of communities in small-world networks and the means to find them. Newman [103] analyzed communities within networks and the means to find them and Chen et al. [32] did similar work concerning evolutionary networks. Krause et al. [85] discussed the conflicting nature of different features in a graph and their alignment. For example in wireless ad hoc systems, routing and access control have opposite effects on the ideal network structure. Milo et al. [98] looks at networks in a more holistic sense, discussing what usually is represented by nodes and links within a network. Also discussed were various common network motifs, what they represent and appearances of such in nature. Newman et al. [105] discussed various types of random graphs; where they exist in nature, means of calculating important measures for such graphs and applications which use such graphs. Criado et al. [38] discussed various node properties in the context of a nodal leader or most important node. The analysis was similar to some key measures such as centrality, radius and cut vertices, and was also discussed in the sense of attack or failure tolerance. Barat et al. [9] analyzed weighted networks and demonstrated that the weights of different links are often more important to the overall dynamics than the topology alone.

Among other research in the field of networks is that of network growth. Growth often times means that the network is growing in size by specific rules, but it may also be used in cases where the number of links or nodes does not dramatically increase, but still changes over time such as equilibrium graphs. For this reason sometimes it is called graph evolution rather than graph growth. Since structure and properties are important, many people want to know the ways and means to build a network with those properties. The most famous or classical examples of network growth or formation are the Erdos-Renyi random graph, the Watts-Strogatz model, the exponential graph and Barabasi-Albert model for the scale free networks by preferential attachment.

The first of the four in terms of publication was the random graph studies by Erdos and Renyi [45] [46] . Their work extensively analyzed graphs having N vertices (labeled such that structurally identically graphs are allowed) and n edges. The graph is formed by selecting n edges at random out of the possible $\frac{n(n-1)}{2}$ total edges. Another model is of the form N, p, where once again N is the total number of nodes, and p is the likelihood that any node pair will share an edge (each edge is independent from the rest).

The models are linked by the equation, $= p\frac{N(N-1)}{2}$, but in the second case the number of edges is not assured to be n. In general the second case is more often used due to the independence of each node (and that model is used for the results given below).

The research focused on four main questions:

- What is the probability of the graph being completely connected?
- What is the probability that the greatest connect component of the graph has n-k nodes (where k is a non-negative integer)
- What is the probability that the graph should consist of exactly k+1 connected components (k is a non-negative integer)
- If the edges of a graph with n vertices are chosen successively so that after each step every edge which has not yet been chosen has the same probability to be chosen next, and if this process continues until the graph becomes completely connected, what is the probability that the number of necessary steps v will be equal to a given number l.

And despite finding results for each of these as the number of nodes tend to infinity, only a few results are of interest for graphs of finite size. Those results serve as thresholds for graph behavior based on the values of n and p and those thresholds are given below in Equation 5 [46] :

Equation 5 Connectivity Thresholds for ER Networks

$$p = \frac{\ln \mathrm{n}}{N}$$

$$Np = 1$$

If the probability p is greater than the threshold given above the above equation, the graph will almost surely be connected; conversely the graph will almost surely be disconnected for a value of p below that threshold, due to isolated vertices. For the second threshold, a graph having np less than one will almost surely have no connect components greater than O(log(N)) in size. If Np is equal to one the largest component will be of size O($N^{2/3}$). For Np greater than 1 there will almost surely be one giant component, and no other components will be greater than size O(log(N)). Some criticisms of the Erdos-Renyi graph model come from their difference with models found in nature; namely that such models have a low clustering coefficient and do not contain largely connected hubs. An example of an Erdos Renyi graph with colored components based on size is given below in Figure 25.

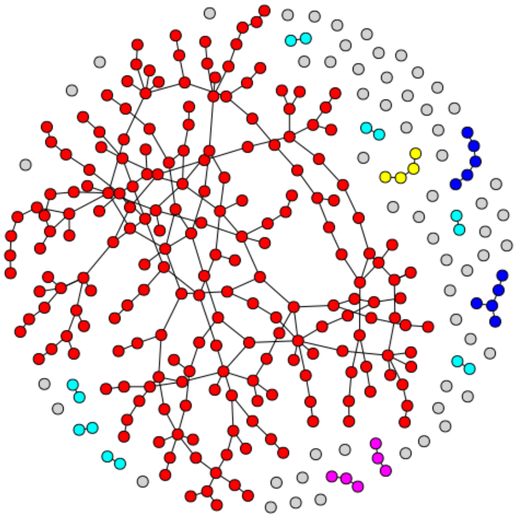


Figure 25 Example Erdos-Renyi Random Graph [71]

After these results were found, Watts and Strogatz [141] aimed to create another random graph, but this time to have a higher connectivity as compared to the Erdos-Renyi random graph. This results in the graph having the "small-world" property. The graph is formed by starting with a ring graph where each node is connected to its K closest neighbors (K is assumed to be an even constant) K/2 on each side. The nodes are labeled from 0,1,…N along the graph. At this point, the graph has NK/2 total links and is not highly connected, but that will be taken care of next. The next step in forming the graph is to start on a node and will probability β for each link connecting to a higher numbered node, rewire that link. Links that are rewired have an equal probability of being instead connected to any other node in the graph such that no link is repeated and no loops are formed. Clearly the value of beta has a great effect on the network, and with a value of one, the graph will become the Erdos Renyi random graph. The most important characteristics of the Watts-Strogatz graph are the average path length, and clustering coefficient. For beta values of zero, the average path length is $\frac{N}{2K}$ and rapidly approaches $\frac{\ln N}{\ln K}$ as beta approaches one. This clearly shows that the average path length is greatly reduced by the rewiring procedure. Conversely, it can be shown that the clustering coefficient is proportional to $(1-\beta)^3$ so a low beta value is advantageous for clustering of the graph. These results combined are the reason why low beta values, but not too close to zero from Watts-Strogatz models with the most "ideal" conditions in terms of the above metrics. Figure 26 below shows how the graph changes with different values of beta.

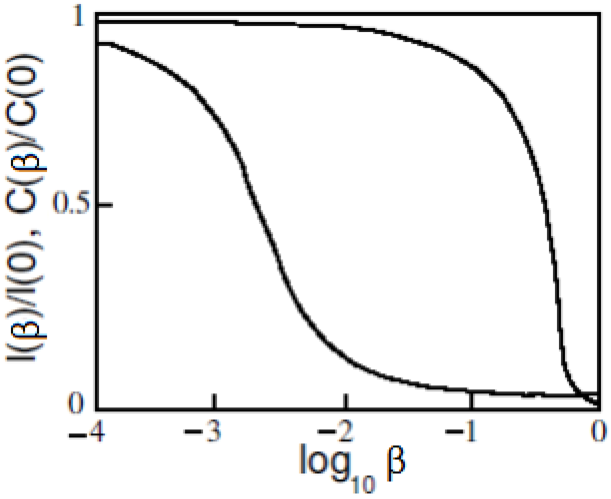


Figure 26 Example of change of normalized average path length (l) and clustering coefficient (c). Network has 1000 nodes with K=10 [42]

Similarly to the Erdos Renyi graph, common criticisms of the Watts-Strogatz model are its lack of highly connected hubs, which makes it unsuitable for representing some systems. In addition to this technique of rewiring, another model was created by Newman and Watts [106] which added "shortcut" links instead of rewiring them with similar results. An example of a Watts-Strogatz graph is given below in Figure 27.

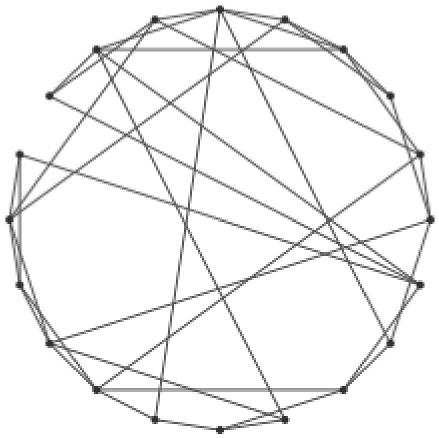


Figure 27 Example Watts-Strogatz Small World Network [60]

The next model of graph growth is the exponential model. It is a method for growing networks similar to the Erdos-Renyi random graph, but the end result is a bit different. These graphs start with two nodes connected to each other by a link. From here for each time step a new node is added and it is linked with one existing node at random and with no preference. If the initial state of the graph is at time one, then at time t, there will be a total of t+1 nodes with t links. Similarly to the Erdos-Renyi random graph, the average path length is proportional to the natural log of t (i.e. the number of nodes in the network). Such networks are called exponential because their degree distribution has an exponential form, which contrasts the Poisson degree distribution of Erdos-Renyi random graphs. Unlike the Erdos-Renyi random graph, the flexibility of these graphs is quite minimal; in all cases the graph is connected since it starts connected and each new node is then linked to that connected component. In addition, even though each link is added completely at random, the final graph has correlations between nodes favoring the older nodes in the graph. This makes sense because the oldest nodes have the most opportunity to be linked to new nodes. These graphs also lack the highly connected hubs found in many real world networks despite the fact that the early nodes have a higher chance of having a greater degree. It also lacks a large clustering coefficient and therefore lacks the small world property. Variations on this model include the ability to add multiple links from each node as it is formed, but the basic properties remain essentially the same in concept.

The final “classical” example of graph growth is the Barabasi-Albert [8] model used to create scale free networks via preferential attachment. As mentioned above, in works on scale free networks, Barabasi found that many real world networks are scale free and considered ways to grow a network beyond the traditional means and have that network maintain the scale free property. It is interesting to note that Barabasi et al. was not the first researcher to use the idea of preferential attachment, but their work popularized and spread the idea, and therefore it is attributed to them. The simplest and most effective way to do so is to use the idea of preferential attachment. Preferential attachment means that each new node has a larger chance of connecting to an existing node with a large degree rather than connecting to an existing node with a small degree. Specifically, the chance of a new node connecting to node i is given by $p_i = \frac{d_i}{\sum_j d_j}$. This creates a system where hubs tend to form, as large degree nodes tend to gain even larger degrees. Interesting properties of graphs formed via this method are the average path length and clustering coefficient. The average path length is proportional to $\frac{\ln N}{\ln \ln N}$ which is shorter than that of a random graph. The clustering coefficient cannot be found analytically, but it approximately follows a power law based on the number of nodes given by $C \sim N^{-.75}$. One thing to note about graphs formed this way is that the preferential attachment as well as the growth is essential. By taking a graph with fixed size and adding links via preferential attachment, the graph does not become scale free. Due to the popularity of scale free networks, many models have been made to produce them with different results. However, the perceived commonality of this type of graph in literature can be deceiving, as oftentimes only a portion of the network is scale free. An example of a scale-free network is given below in Figure 28.

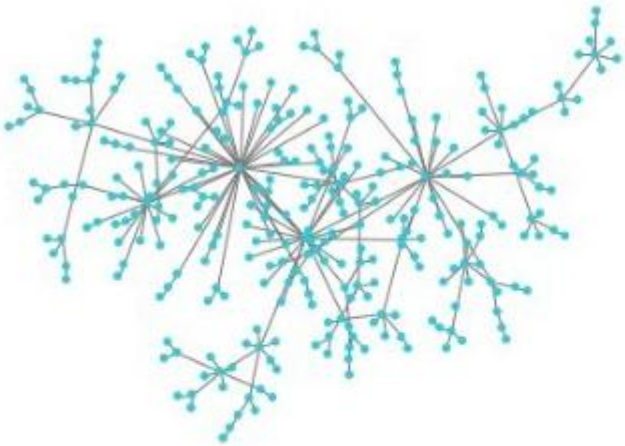


Figure 28 Example of a Scale Free Graph [20]

Other examples of graph growth are less widespread but may be of greater applicability to a given area. Riccaboni and Schiavo [119] extended the model of preferential attachment to include randomness in the vein of random Brownian motion for weighted networks. Bornholdt and Rohlf [15] created a model of evolution with local dynamics that produced new and unpredicted global trends. This is like the concept of emergence commonly discussed in complex systems analysis. Chan et al. [28] created a model of random addition of nodes and links into an existing graph. Different types of graphs were created, with major types having the “small-world” property, lesser linked graphs and graph trees. Jensen [73] created an evolutionary model of a biological sense with the death of nodes and the birth of nodes which may have similar links to their parents and also with possible mutations. Newman [102] discussed the idea of assertive mixing in networks as they evolve. It is a concept similar to preferential attachment in which nodes with a large number of links are likely to create links with other nodes with a large number of links. Also discussed were cases when this property holds and when it does not.

2.2 Cooperative Control

Many definitions exist for cooperative control. Fax et al. [47] define cooperative control as "a collection of vehicles performing a shared task using intervehicle communication to coordinate their actions." However, a cooperative control system does not need to involve vehicles specifically. Another definition is provided from Shabab [125] "Cooperative control is a collection of interconnected decision-making components all seeking to achieve a collective global objective." For this research, cooperative control is defined as the individual and group control techniques applied over a distributed system such that each individual agent contributes to the completion of an overall team goal. Examples of systems which use cooperative control are given below:

- Mobile Sensing Networks [36]
- Vehicle Formations [47]
- Multi-Robot Remote Driving [51]
- Multi-Robot Foraging [91]
- Search and Attack UAV Swarms [112]
- UAV Collaborative Sensing [122]

2.2.1 Control Considerations

Perhaps one of the first considerations to make when designing a cooperative control scheme is whether or not the scheme is centralized or decentralized. A centralized scheme is what it sounds like, there exists some central vehicle or authority which collects information from the rest of the team. This information is processed and strategy/decisions are made from this and communicated to the rest of the team. A decentralized scheme is one which each vehicle operates on the local information it has to complete the overall team goal. This local information may contain information directly sensed or detected by the vehicle, in addition to information communicated to it by its neighbors. Advantages and disadvantages exist for both cases. Centralized control has the main benefit of team agreement on decisions because a single entity is making the decision for the entire team. As long as the "commanding" entity is logical, this will ensure the team is operating toward the same goal, rather than on different actions based on differences in local information. The downside of this method is that the leader can represent a single point of failure. Also, the amount of time it takes to collect and redistribute tasks and decisions can make the system less agile than alternative methods. Conversely, decentralized schemes are more agile due to their ability for each agent to act on its own local information. Any change detected in the system can be acted upon immediately rather than requiring some up time to the central entity than down time for what should be done. However this agility can come at the cost of disagreement between members of the autonomous team. Local information may cause differences in knowledge between individuals, causing them to believe differently as to what should be done, meaning an overall mixed strategy. This can cause reduced performance of the desired mission, because parts of the team may be acting counter to the rest.

Regardless of the above choice made, communication and the agreement of information is a critical component to the success of the overall mission. It is desired that all members of the team have as accurate and complete an idea of the overall system state as possible. This is not possible usually do to issues of communication lag, as well as communication networks which may cause information flow to be sluggish. In general, the communication network is a critical component to the control, and limitations are placed on it to ensure that the control will operate effectively. Ideally, the network will be fully connected, allowing for the most rapid flow of information throughout the system [35] . This will not always be the case, and when the network is not fully connected some agreement methods need to be implement to alleviate a non-ideal situation. One of the most common agreement techniques is consensus [117] , in which information is shared and modified until each member is in agreement.

Another consideration is the means of which control is dictated. Many classical control schemes, such as linear control, use differential equations to dictate state changes based on some rules. This is also the case for some cooperative control systems. A prime example is that of vehicle formation control. As the formation encounters perturbations from the environment, vehicles may me moved outside of their desired state, and their future movement will be based on these laws. An example of such is given below in Equation 6.

Equation 6 Sample Formation Control

$$\dot{x}_i = \sum g(x_i - x_j)$$

These types of techniques are used mainly in vehicle dynamic problems, however these types of problems may also be solved via other means. Optimization techniques are also popular in the realm of cooperative control. Some formation problems use this as well. Dunbar and Murray [43] created a cost function which considers both vehicle position as well as communication. In general a more tight formation criteria can be maintained by increasing the need of communication, so this technique allows a way to trade off the two issues. Other optimization problems exist in the realm of optimal sensor coverage [36] of vehicles around a target. These techniques are advantages because they allow a team of disparate entities to try and achieve the same goal. Reconfiguring based on a similar set of rules can allow for an easy way to transition the system when information is in agreement. Other more specialized techniques exist, such as using market based assessment to determine task objectives [29] , as well as using gradient based approaches to dictate vehicle motion [121] .

2.2.2 Summary of Existing Control Problems

As can be expected from complex systems such as these, very few fundamental ideas or theories are found for such control systems. In most cases the control behavior determined can be used for that problem only, and may have little to no contribution to other outside problems [5] . However the problems that have been studied do yield interesting ideas if not direct applicability to other such problem. Murray [101] gives a good overview as to some of the applications of cooperative control and covers some classic problems in the field such as formation control, rendezvous, coverage etc. Cortes et al. [36] assumed full communication and made mathematical simplifications of vehicle dynamics and sensor functions in order to create a mathematical formulation of optimum vehicle placement for a sensing mission. They found that the best locations for each vehicle corresponded to the centroid of an optimal Voronoi diagram created. Fax and Murrary [47] analyzed ways to coordinate vehicle formation with and without communication (sensors to determine location of other vehicles were used) using eigenvalue analysis. Price and Lamont [112] simplified vehicle functions to mathematical ones, and used those along with a genetic algorithm to determine the best set of behaviors to use to achieve a goal. Many other examples exist in this vein of control of vehicle motion.

Chandler et al wrote two interesting studies for UAVs and cooperative control. The first [29] focused on target classification, but included search, attack and damage assessment in the study. A few interesting components of the study were the means of cooperation via market based assessment. With this idea, vehicles would buy or sell certain task objectives, making it easier to differentiate what vehicle or team would perform what task. In addition, multiple vehicle behavior types were classified and assigned at the beginning of the mission. In addition communication was assumed to be instantaneous with full communication among all vehicles. The second study [30] extended this to more complex missions in which multiple assignments had to be made for task completion. Examples include different vehicles being used to validate a target classification, or to verify damage assessment. Fax and Murray [47] did a study for vehicle formation flying in which communication was not assumed to be instantaneous and complete among all vehicles. In this case a leader was assigned for the network to deliver instructions and maintain consensus. In cases when communication was not possible, vehicle to vehicle sensors were used to maintain formation. Flint et al [50] also did a study in which communication was limited. In this case a group of UAVs whose mission is to search for targets with some a priori information available is studied. To demonstrate communication losses, in different fractions of time steps communication was allowed. Results showed that for lower frequency of communication the overall behavior suffered, but not as much as when no cooperative schemes were used. Ogren et al [107] did a study in which vehicles had two goals: to maintain formation and follow noisy information gradients as a mathematical abstraction of a surveillance mission. Full and instantaneous communication was assumed. In order to best satisfy both mission objectives, the two goals were decoupled.

2.2.3 Multiagent Competitions

In addition to these types of problems are examples of autonomous teams which compete in different tasks with other autonomous teams. Perhaps two of the better examples of this type of competition are the RoboCup [81] and RoboFlag [115] competition. The RoboCup competition was inspired by the soccer world cup, and involves two teams of autonomous robots playing a form of soccer against each other. This contest is interesting because it seeks to find improvements in robotic mechanical ability, low level dynamics as well as strategy. Improvements in any category can make the team better and more likely to succeed. The goal of this competition is to improve the capabilities of robots to perform activities on equal ground to human counterparts.

The RoboFlag competition is one in which two teams of autonomous vehicles compete in a capture the flag type of game. This problem is different than the RoboCup competition because both teams have equal capabilities. In this case, it is only the choices of strategy which can cause a given team to lose or win. While low level control exists in this problem, it is the modification of high level strategy conditions which dictate victory or defect [116] . Variations exist for this problem in terms of how the overall strategy is chosen and allowed to change; the strategy can be chosen autonomously via some centralized process, or a human may be allowed to serve as a general of sorts and dictate strategy.

Both of these contests require the team to work in less than ideal communication scenarios and less than optimal conditionals, however the problem is still fair because both sides are subject to the same limitations. Problems like these are interesting because it is difficult to determine success or failure for a given control strategy. The problem has some aspects of chess, in that many moves are made while the final determining factor is victory or defeat. This does not mean that any move made in a victory was a good or advantageous one, or any move made in defeat was a bad choice. The above problems are made slightly easier because the determination of victory is made by the team which scores more points. Allowing a point implies some part of the strategy was poor, while scoring one implies some part of the strategy was good. However, it does necessarily imply that all the actions between allowing or scoring a point are good or bad.

The difficulty of determining the correct strategies and when to implement them practically necessitates the testing of such methods in the field or via simulation of competition. It is not possible to easily determine when or how to change strategies based on the situation with limited knowledge and other constraints. This difficulty is what makes these problems unique and interesting to study vs. cooperative control problems in more static situations. In a static situation, use of some mathematical formalization can give the best strategy at a high and low level due to the more strict limitations set on the overall problem.

2.2.4 Role of Communication Network in Control

There are many papers which consider how to control a network for various means. There is a lack of papers which attempt to look at the problem where the network itself cannot be controlled, but control has to be applied across the network. Even dynamic networks which use agreement techniques have requirements on the network such that it be connected at all times[142] [143] . Clauset et al. [35] attempted to do this and found some similar results, but some new as well. They chose the three most important network properties to be connectedness (as above), but also navigability and efficiency. Navigability in this case applies to the ease of finding a path between two nodes of the network, and the longer that takes the harder it is to spread information to needed areas of the graph. In true problem dependent fashion, efficiency is a problem dependent measure, determining cost of navigation of the network. This paper did not actually have results as the problem was difficult and instead ended with some areas of future research. With that considered many researches readily accept the relative lack of maturity of the field [74] [5] .

A network having the best properties for cooperative control is a highly researched area, with many researches looking into what are advantageous features for a network to have and how to design it properly to both perform well and be robust to losses, as well as other issues. Hoveareshti [68] discusses ways to make the communication network such that it has good consensus formation as well as be robust to some loss of communication or error. Zavlanos and Pappas [144] discussed methods to insure proper connectedness throughout the activity of the network of entities such that information would always be able to flow through the entire process. Goodwin et al [59] analyzed the effect of actual communication losses on a network, such as packet loss and measured their effects on the overall performance. Other authors changed the performance or activity of the entities based on conditions within the system. Hsiegh et al. [69] describe a system in which the communication between vehicles throttles up and down depending on mission performance.

2.2.5 Consensus and Related Issues

Agreement of information among entities within the network is key in cooperative control systems. Centralized schemes or those with fully connected communication networks alleviate the need for agreement because the transfer of information takes a minimal amount of communication to achieve agreement. However, these systems are problematic in the real world because centralized schemes have single points of failure and fully connected communication networks may not be possible given the mission parameters. The information required for coordination is called coordination information or coordination variable [95] . If a cooperative control scheme is effective when such information is globally known by each entity, cooperation would occur. In cases of unreliable, or changing information and communication topology each vehicle will be operating on different information. A means of establishing consensus of vehicles in these cases is essential for effective control. Consensus algorithms exist for various types of problems, but the main distinction is between continuous and discrete timescales. In a basic sense, both techniques give an averaging scheme based on network topology which is devised to help convergence off the coordination information among vehicles when possible. A good source for issues of consensus can be found in the work of Ren et al. [118] . An example of consensus being reached in a group of systems is given below in Figure 29.

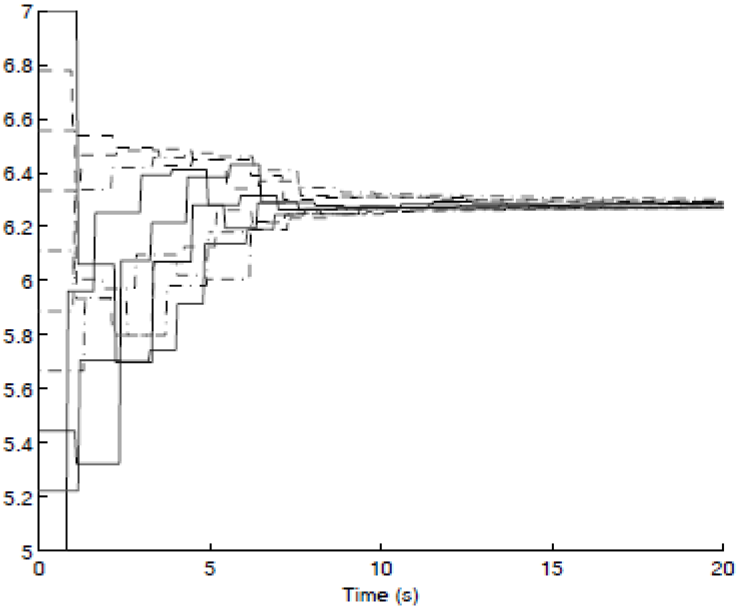


Figure 29 Example of Consensus over time, modified from [117]

In the continuous case, the updating mechanism for the coordination information (ξ) is given below by Equation 7.

Equation 7 Consensus Updating Mechanism

$$\dot{\xi} = -[L_n(t) \otimes I_m]\xi$$

$$\xi = [\xi_1^T, \dots, \xi_n^T]^T$$

Where ξ is a 1 x (nm) column vector which is formed by placing each individual coordination vector in a single statewide coordination vector, $L_n(t)$ is the n x n Laplacian matrix at time t, I_m is the m x m identity matrix and ⊗ is the Kronecker product. And consensus is achieved when the condition given in Equation 8 is reached. Equation 8 is given below.

Equation 8 Consensus Criteria

$$\left\|\xi_i(t) - \xi_j(t)\right\| = 0 \qquad for\ i, j = 1,2, \dots, n$$

This is to say that each coordination vector is equal. For static communication topologies a few matrix properties are necessary for consensus to be reached. For an average consensus to occur the communication graph must be fully connected for undirected graphs, or strongly connected for directed graphs. The exact average will occur in balanced networks, while a weighted average will occur in non balanced networks. That is equivalent to the following statements:

- L_n has a simple zero eigenvalue with associated eigenvector [1,1,…,1] and all other eigenvalues are positive and real (or positive real parts for directed graphs)
- The rank of L_n is n-1

There is an additional allowance for consensus to be reached, however it is not a weighted average of all coordination vectors, but rather a distribution of one coordination vector to the rest asymptotically. It can only occur in directed graphs, when node k has zero in degree (note only one node can have this property). It is equivalent to the following statements:

- The communication network is weakly connected
- The k-th row of L_n has zero for every entry

If neither of these conditions is met, the system cannot reach consensus. It is also possible for the network to be dynamically switching, which is to say that a set rule changes from one topology to another through a set amount in a cyclic manor. For these systems, the overall behavior of system is essentially slower than that of the static system [142] [143] , and the overlapped graph which is the combination of each individual graph dictates whether it can reach consensus or not. In some strict cases of graph switching in which the nodal degree is kept constant, convergence may be increased due to a more rapid mixing like process of the information between the members of the network [117] .

The discrete case is one in which the updates occur at distinct time steps. It is useful in communication schemes were communication occurs in intervals rather than continuously. The updating scheme for the coordination vector (ξ) is given below by Equation 9.

Equation 9 Consensus Information Updating

$$\xi[k+1] = (D_n(k) \otimes I_m)\xi[k]$$

In this case D_n is a stochastic row matrix (each row sums to one). For this case the values of D_{nij} are taken to be the in degree of node i from node j and D_{nii} is 1 (each row must be normalized to sum to one). Essentially this is a weighted average of each nodes coordination vector with each incoming coordination vector based on communication network structure. This is an extension of the simple updating technique given in [72] in which the Laplacian matrix is replaced by a matrix defined by a_{ij} = 1/(1+m) when node i is connected to j and a_{ij} = 0 otherwise (except a_{ii} which is defined to be 1/(1+m)). In this case m corresponds to the in degree of node i. This is to say that the updated coordination vector is a simple average of its current value and all received values. This technique suffers from slower convergence rates, but does have the advantage of not depending on each vehicle knowing the total network topology, should it be changing. The requirements for a consensus to be reached are the same conditions which must be met for the continuous case in terms of connectivity and related issues.

Many classic consensus applications exist. A few of those are: rendezvous and alignment with multiple wheeled robots; distributed formation control with a virtual leader; decentralized behavior approach to mobile robot formation maneuvers; deep space spacecraft formation flying; cooperative fire monitoring with multiple UAVs and cooperative surveillance with multiple UAVs.

2.3 Assignment Optimization

Optimization in cooperative control is often one of assignment. In the most basic sense, assignment takes n assigners and n assignees and finds a combination or permutation which assigns each of the assigners to a single assignee. Burkard et al. [21] gives a good background to assignment problems. Figure 30 below gives a sample assignment.

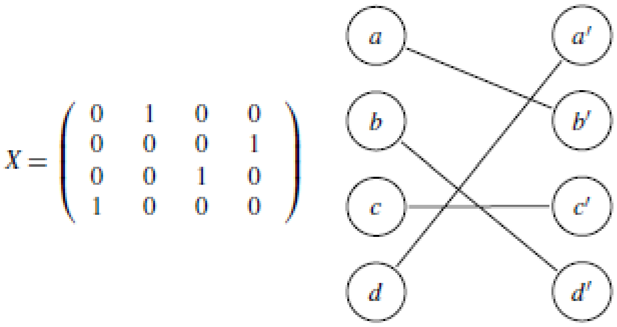


Figure 30 Example of an Assignment. Modified from [21]

In the basic case, each assigner can be assigned to any assignee. For that case, the assignment matrix can have many permutations which must satisfy the following conditions and constraints given in Equation 10.

Equation 10 Standard Assignment Constraints

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$x_{ij} \in \{0,1\} \; for \; all \; i,j = 1,2,\dots,n$$

This ensures that each assigner and each assignee have one and only one partner. This is the most basic type of assignment problem and there are many different types of assignment problems which fall into three basic categories; linear assignment problems; quadratic assignment problems and nonlinear assignment problems. Each will be covered with some basic problems in the space and techniques of solving them. One common trait of many assignment problems (especially the quadratic assignment problem and nonlinear assignment problem) is that finding the exact or unique minimum is quite difficult in that it takes a long time. However heuristic techniques and approximations can be made which give relatively optimal solutions in much faster time.

2.3.1 Linear Assignment

The case mentioned above is a linear assignment problem of the most basic class. Without any additional information any assignment obeying the constraints will work, it is simply a matter of assigning n pairs without pair preference. Another problem which exists in linear assignment problems is when some pair wise matches are not allowed. Another extension of the classic problem takes this into account by introducing the bipartite graph G which essentially states what pairs are allowed by having $g_{i,j}$ equal to one, and what pairs are not allows when that number is zero. The problem in this case is to find if a set of pairs such that each node has a partner. Early in the development of this problem, it was explained with one group being men and one group being women, and the pair symbolized marriage, and for this reason such problems as this are often called marriage problems. In the case when the men and women have preferences amongst those in the other set, the problem becomes one of stable marriage; which is to say that in each marriage, there is no pair of man or woman who *both* prefer another partner over the one they have. In addition, maximum matching is an important issue in which there is a maximum number of matches exist. The optimum solution will itself be a maximum match, but the converse is not true. Many bipartite matching algorithms exist to solve such problems. The oldest method is one of labeling which is used to find augmented paths which lead to the optimal problem. The Hopcroft-Karp algorithm [66] augments this procedure to allow for better speed of optimization via augmentation of a shortest augmented path. Alt et al. [3] improve upon this method for dense graphs using fast adjacency scanning techniques developed by Cheriyan et al. [33] . Glover [56] [57] developed an efficient means of maximum matching in convex bipartite graphs. A convex bipartite graph is one which can be rearranged such that for one set of nodes, all of its neighbors fall into a consecutive complete range. IE if node one is connected to nodes four and five that is a consecutive range for a single node. If node one is connected to node four and seven, that is not. Some examples of maximum matching problems are: vehicle scheduling problems and time slot assignment problems.

Linear assignment problems are often of the classic type listed above but with one key difference. Implicit in the formulation was that each pairing was just as good or bad as any other. Oftentimes that is not the case, and some pairs may be more advantageous than others. Mathematically this is accomplished by using a cost matrix C, in which each $c_{i,j}$ contains the cost of that pair wise assignment. In such a case the new optimization problem becomes what is given below in Equation 11 and Equation 12 (minimization and maximization problems can easily be exchanged).

Equation 11 Linear Assignment Problem

$$\min \left[\sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{ij} \right]$$

Equation 12 Associated Linear Assignment Constraints

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$x_{ij} \in \{0,1\} \;\; for\ all\ i,j = 1,2,\dots,n$$

Now the problem becomes one of finding the best combination of pair wise matches rather than using any old matching scheme. This problem is called the Linear Sum Assignment Problem (LSAP). One of the first solutions to problems of this type was the so called Hungarian algorithm reported by Kuhn [87] . Kuhn called it the “Hungarian Method” based on Konig’s classical graph theory book [82] and one of its references by Egervary [44] which Kuhn personally translated from Hungarian. Given n workers and n tasks, and n x n matrix is created for which each row (i) represents the costs for worker (i) to do the task (j) corresponding to column (j). The first step of this method is to find the smallest element for each row and subtract it from the remaining elements in that row. This will give at least one zero valued element in each row. If this step gives n zeros corresponding to one independent zero for each row and column, the index of each zero represents the optimal assignment and the problem is done. If there is no feasible combination (one or more tasks are left unassigned), then the next step of the procedure is to apply the same procedure as step one, except on each column rather than each row. By doing this at least one zero will be in each column. Once again, if a feasible solution now exists, the elements of the matrix with zero value chosen correspond to the optimal matching. If there is not a solution at this stage, then exclude any column (job) which has a unique minimal performer. Of the remaining rows, any column that has multiple options for a performer must also be excluded for the time being. Of the remaining elements, subtract the lowest element from each row, and add it to each element which is removed from consideration by both a row and column removal. If the remaining system yields a solution, the task is finished. If not return to step one and repeat until a solution is found. What this technique does it to find the best task for each performer, then for cases of confliction (ex if both performer 1 and 2 would be best served by performing job 1) it finds the next best job for any conflicting performers and assigns those jobs accordingly. This algorithm was improved by Jonker and Volgenant [75] . A great number of modifications and new techniques exist to solve this problem for various matrix types, preconditioning and other techniques. A good comparison of these methods can be found on page 128 of [21] . One additional item to note regarding the LSAP problem is a variation known as the bottle neck problem. In this variation the sum of each cost is replaced by the largest cost job. In this case the actual function being optimized is given below (in Equation 13), with the remaining constraints from the traditional problem still in place.

Equation 13 Additional Constraint of Bottle Neck Problem

$$\min\left(\max c_{i,j}\, x_{ij}\right)$$

These problems are often of use if the cost is a time variable and the importance is to find the minimum time it would take for all the jobs to be performed.

A few interesting variants to the traditional LSAP exist. Those are: determination of the K best solutions; the k cardinality problem; the semi-assignment problem and the assignment problem regarding rectangular matrices. In some cases of assignment, detailed problem specifics and real world issues cannot be implemented in a linear scheme. Therefore, the best K solutions are chosen and given to decision makers who may decide on the best one considering those added issues. The first solution to this problem was proposed by Murty [101] . Later improvements were made by Chegireddy and Hamacher [31] over Murty's scheme, and finally by Pascoal et al. [110] . The k-cardinality assignment problem reduces the problem to assigning k rows to k different columns such that the corresponding cost is minimized. Specifically the problem is given below by Equation 14 and Equation 15.

Equation 14 K-Cardinality Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}\,x_{ij}\right]$$

Equation 15 Associated Constraints

$$\sum_{j=1}^{n} x_{ij} \le 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{j=1}^{n} x_{ij} \le 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \ = k$$

$$x_{ij} \in \{0,1\} \ for\ all\ i,j = 1,2,\dots,n$$

The first algorithm to do this problem was devised by Dell'Amico and Martello [39] . It was found that this problem could be solved using any linear programming solver due to the nature of the added constraint. However, preprocessing techniques would not be as applicable, though those were also developed by Dell'Amico and Martello [39] . Additional efficient implementations were created for both sparse and dense cost matrices by Dell'Amico and Martello [39] . In addition, Volgenant [140] described a technique to transform problems of this type into standard linear sum assignment problems.

The semi-assignment problem involves an n x m cost matrix (n > m) and a vector b of m values. For each row, one element must be selected such that the number of entities performing each job (j) is equal to b_j. For this formulation to work, the sum of the elements in b must sum to n. A mathematical formulation is given below in Equation 16 and Equation 17.

Equation 16 Semi Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}\, x_{ij}\right]$$

Equation 17 Associated Constraints

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{i=1}^{n} x_{ij} = b_j \qquad (j = 1,2,\dots,m)$$

$$x_{ij} \in \{0,1\} \; for\ all\ i,j = 1,2,\dots,n$$

Barr et al. [10] give an adaptation of their alternating basis algorithm to solve this problem. Kennington and Wang [80] devolved an algorithm using results from [75] that solves large size sparse and dense instances of the problem. The final problem is that of the rectangular cost matrix of size n x m where n < m. This problem can be easily solved by adding dummy rows to the cost matrix of zero valued elements and solving the problem with other techniques. Specialized algorithms for this problem were developed by Bourgeois and Lassalle [16] [17] . Examples applications using linear cost assignment are: mean flow time minimization [67] ; categorized assignment scheduling [113] ; optimal depletion of inventory [40] ; personnel assignment with seniority and job priority [26] ; Navy personnel planning [65] among others.

2.3.2 Quadratic Assignment

The quadratic assignment problem is an extension of the linear assignment problem. Perhaps the easiest to understand formulation of this problem is the Koopmans-Beckham formulation [83] , which is given below in Equation 18 and Equation 19.

Equation 18 Quadratic Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{i,j}\, b_{k,l} x_{i,k} x_{j,l}\right]$$

Equation 19 Associated Constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1,2,\dots,n)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$x_{ij} \in \{0,1\}\ for\ all\ i,j = 1,2,\dots,n$$

Where A ($a_{i,j}$) and B ($b_{i,j}$) are cost matrices of size n x n and X ($x_{i,j}$) is a permutation matrix of size n x n. This formulation essentially minimizes the relative cost between each pair of assignments. If at least one cost matrix A, B is symmetric, then the problem is called the symmetric quadratic assignment problem. A matrix M ($m_{i,j}$) satisfies the triangle inequality if the following holds for all indices (i,j,k) given below in Equation 20.

Equation 20 Triangle Inequality

$$m_{i,j} \leq m_{i,k} + m_{k,j}$$

If this holds that matrix is said to be Euclidian. If at least one matrix A,B fulfills the triangle inequality (is Euclidean), it is said that the quadratic assignment problem of A,B fulfills the triangle inequality (is Euclidean). Some applications of QAP are facility location problems [83] , scheduling [54] , wiring electronics problems [133] , parallel and distributed computing [14] , statistical data analysis [23] , archeology [61] , [76] , chemistry [137] , [52] etc. Finding exact solutions to problems of this type are usually done with difficulty by using Branch and Bound methods. This is an inefficient means to achieve the exact optimum due to the unpredictable nature of the algorithm. Other problems in this area are of linearization, lower bounds for the exact problem, and heuristic approaches.

Linearization is a common method for solutions due to the difficulty of the quadratic nature of the objective function. The most successful linearization schemes are mixed integer linear programs (MILP), although they create a large number of additional variables and equations makes the problem still difficult to solve. The two linearization techniques discussed are the Kaufman and Broeckx [77] linearization and the Frieze and Yadegar [53] linearization. The Kaufman Broeckx linearization is likely the smallest linearization as it adds n^2 Boolean variables, n^2 real variables and n^2+2 constraints. The added variables introduced into the system are given in Equation 21.

Equation 21 Added Variables from Kaufman Broeckx Linearization

$$y_{ik} = x_{ik} \sum_{j=1}^{n} \sum_{l=1}^{n} a_{ij} b_{kl} x_{jl}$$

$$d_{ik} = \sum_{j=1}^{n} \sum_{l=1}^{n} a_{ij} b_{kl}$$

And the new linear formulization is given below Equation 22 and Equation 23.

Equation 22 Kaufman Broeckx Linearization Assignment Problem

$$\min \left[\sum_{i=1}^{n} \sum_{k=1}^{n} y_{ik} \right]$$

Equation 23 Associated Constraints

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad (k = 1,2,\dots,n)$$

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad (i = 1,2,\dots,n)$$

$$d_{ik}x_{ik} + \sum_{j=1}^{n}\sum_{l=1}^{n} a_{ij}b_{kl}x_{jl} - y_{ik} \leq d_{ik} \qquad (i,k = 1,2,\dots,n)$$

$$x_{ik} \in \{0,1\}, \;\; y_{ik} \geq 0 \qquad (i,k = 1,2,\dots,n)$$

Despite the ease which is brought on by the linearization, other problems arise due to the added complexity from variables and constraints. "…even this linearization which perhaps the smallest one, has a large number of variables and constraints. Under these conditions even powerful tools to cope with integer programs…do not help a lot. It turns out that for QAPs arising in practical applications even solving the relaxed linear program is computationally a hard job." [27]

The Frieze and Yadegar [53] linearization technique is also MILP and adds n^4 extra real variables defined below in Equation 24.

Equation 24 Added Variables from Frieze Yadegar Linearization

$$y_{ijkl} = x_{ik}x_{jl} \qquad (i,j,k,l = 1,2,\dots,n)$$

That along with n^2 Boolean variables and $n^4+4n^3+n^2+2n$ constraints gives the following MILP problem given below in Equation 25 and Equation 26.

Equation 25 Frieze Yadegar Linearized Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{i,j}\, b_{k,l} y_{ijkl}\right]$$

Equation 26 Associated Constraints

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad (k = 1,2,\dots,n)$$

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad (i = 1,2,\dots,n)$$

$$\sum_{i=1}^{n} y_{ijkl} = x_{jl} \qquad (j,k,l = 1,2,\dots,n)$$

$$\sum_{j=1}^{n} y_{ijkl} = x_{ik} \qquad (i,k,l = 1,2,\dots,n)$$

$$\sum_{k=1}^{n} y_{ijkl} = x_{jl} \qquad (i,j,l = 1,2,\dots,n)$$

$$\sum_{l=1}^{n} y_{ijkl} = x_{ik} \qquad (i,j,k = 1,2,\dots,n)$$

$$x_{ij}\,, y_{ijkl} \in \{0,1\},\;\; y_{iikk} = x_{ik} \;\; for\ all\ i,k = 1,2,\dots,n$$

This technique is used to derive a lower bound on the QAP by solving a Lagrangian relaxation of it. Providing bounds on the QAP is an effective means to improve the performance of Branch and Bound techniques used for finding the exact optimum. A good survey on Branch and Bound methods can be found in [89] . There are five main types of lower bounds used today: Gilmore-Lawler lower bounds; eigenvalue related lower bounds; reformulation based bounds; lower bounds based on LP relaxations and lower bounds based on semi definite relations. The main issue of bound techniques is to find the best bounds in the fastest amount of time. As can be expected of such a complex problem, the best bounds are usually had by the most time extensive techniques and vice versa. For general QAPs the dual procedure from Hahn and Grant [62] seems to offer the best mix of quality and time, and is well suited for implementation in branch and bound schemes.

Heuristic techniques are useful because they can give a close to optimal solution in much less time than an extensive optimization. Some effective techniques used to solve QAPs are limited enumeration techniques and tabu search algorithms. Limited enumeration techniques are very similar to the exact techniques used to find an optimum (in this case the branch and bound method), but as the name suggests, it stops the technique before the optimum is found. The idea behind this method is that a solution close enough to an optimal solution can be found during the early stages of the search and the large chunk of the rest of the time is spend proving the optimality of that solution, or slightly improving upon that solution. This behavior can be taken advantage of to provide “good enough” solutions within a more reasonable amount of time. There are two main types of limited enumeration techniques: time limit techniques and bound modification techniques. Time limit techniques set a specified time limit that the technique stops at regardless of what the solution is. They also may include a requirement of sufficient increase in quality of solution, such that for a certain number of new possibilities searched, an increase in quality of x percent must occur for the technique to continue. Bound modification techniques act directly on the bounds found for each node not yet branched during the branch and bound method. The idea is that if no improvements have been made in a specific time interval, the lower bounds are increased by a certain percentage. This may have the consequence of eliminating a possible optimal solution, but because the interest is a good solution rather than the best solution, this is acceptable as it speeds up the search. An interesting note about this specific technique, is that the final solution is known to be bounded below from the actual optimum by an amount based on the largest increased bounds and the smallest lower bound before the increase.

The Tabu search technique is thought by many researchers to be a useful heuristic for solving hard combinatorial optimization problems, specifically the QAP [27] . The technique was first proposed by Glover [56] [57] as a way to overcome local optimality in local search applications applied to combinatorial optimization. The main components of the tabu search are the moves, the tabu list and aspiration criterion. A move is an operation when applied to a solution, gives a new neighbor solution. For QAPs the moves are usually transpositions. A tabu list is one of forbidden moves which cannot be made to the current solution. The aspiration criterion is a condition that when met by a tabu move cancels its tabu status making it allowable. A tabu search procedure starts with a feasible solution, and selects the best quality solution from the neighbors of the current solution which are not tabu moves. This move does not necessarily find a better solution than the previous. The new solution is now the current solution. Due to the possibility of cycles occurring in this method, moves which are believed to yield cyclic behavior are restricted via the tabu list. Making certain moves forbidden may however eliminate search of other solution spaces, and therefore the aspiration criterion are included to make these moves possible. The length of the tabu list itself is of fixed value, and as the tabu list fills up, a first in first out system is used to discard old tabu moves from the list. The tabu search stops based on a limit on run time or a limit on the number of iterations. One of the first tabu search techniques applied to QAP was that of Skorin and Kapov (1990). One of the difficulties in this method was that of tuning of control parameters, which was found to be strongly dependent on the problem instance. This led to poor behavior of the algorithm in terms of robustness. A new version of the tabu search called the robust tabu search was developed by Taillard [135] as a means to make the system more robust to the choice of control parameter value. Another version of the tabu search called the reactive tabu search was developed by Battiti and Tecchiolli [11] which aims at weakening dependencies on the performance by the values of the control parameters. This is done via a mechanism which allows the tabu list to have adapting length; the length of the tabu list increases when the program is believed to be in a cycle. In addition, if solutions are revisited a larger number of times, a random diversification occurs and forces a search towards a new feasible solution. Greedy randomized search algorithms [49] , [93] may be another option in the short term. These techniques seem advantageous in the short run, but in the long run are beaten by other techniques [27] . Other techniques found not to behave as well for general cases as those mentioned above are simulated annealing and genetic algorithm.

2.3.3 Nonlinear Assignment

Nonlinear Assignment Problems (NAP) are perhaps the hardest assignment problems to solve. While QAP is technically a nonlinear assignment problem, it has a great deal of research in of itself and therefore was given its own section. NAP is an extension beyond both LAP and QAP and mainly falls into two areas: multi-index assignment problems (MIAP) and the m-adic assignment problem. The MIAP extends beyond traditional LAP by replacing pair wise combinations with combinations involving three or more items. In the example marriage problem given n men and n women, a MIAP would have n men n women and for example n cats to own as a pet. The problem goes beyond finding the n best pairs to finding the n best triples. This idea can be extended to any number of indices. The m-adic assignment problem (of which QAP is the 2-adic problem) is of less study beyond QAP, which was mentioned in great detail above. For this reason, the discussion of NAP will mainly be to MIAP.

The most immediate extension of the MIAP is the 3-Index assignment problem (3IAP). Within this area there are two problem types, the axial and planar problem. For the axial problem, if there are three sets (A_1,A_2,A_3) of size n, and a weight corresponding to each triple in A_1 x A_2 x A_3 the problem is to find the best combination which yields n triples (one from each set) which minimizes total cost. The mathematical formulation is given below in Equation 27 and Equation 28.

Equation 27 Non-Linear Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} w_{ijk}\, x_{ijk}\right]$$

Equation 28 Associated Constraints

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} \leq 1 \qquad (k = 1,2,\dots,n)$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} \leq 1 \qquad (j = 1,2,\dots,n)$$

$$\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} \leq 1 \qquad (i = 1,2,\dots,n)$$

$$x_{ijk} \in \{0,1\} \; for\; all\; i,j,k = 1,2,\dots,n$$

This type of problem was introduced by Karp [76] . The difficulty of solving the problem means that in general the branch and bound method is commonly used for exact solution. Due to the performance of branch and bound methods, approximation techniques are common. One note in this area is the p approximation, in which the solution is at worse a p fraction of the optimal for the maximization problem. One of the best approximations was developed by Arkin and Hassin [6] to find a solution which is about ½ approximate for the equivalent maximization problem. The A3IAP has many real world applications [111] such as capital investment, dynamic facility allocation, satellite launching and assembly of circuit boards [37] .

The planar problem (P3IAP) is formulated similarly to the A3IAP. For three sets (A_1,A_2,A_3) of size n, and for each triple in A_1 x A_2 x A_3 a number p_{ijk} is known. The problem is to find n2 triples such that each pair of elements from (A_1 x A_2) ∪ (A_1 x A_3) ∪ (A_2 x A_3) is in exactly one triple. The mathematical formulation is given below in Equation 29 and Equation 30.

Equation 29 P3IAP Assignment Problem

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} p_{ijk}\, x_{ijk}\right]$$

Equation 30 Associated Constraints

$$\sum_{i=1}^{n} x_{ijk} = 1 \qquad (j, k = 1,2, \dots, n)$$

$$\sum_{j=1}^{n} x_{ijk} = 1 \qquad (i, k = 1,2, \dots, n)$$

$$\sum_{k=1}^{n} x_{ijk} = 1 \qquad (i, j = 1,2, \dots, n)$$

$$x_{ijk} \in \{0,1\} \; for\ all\ i, j, k = 1,2, \dots, n$$

P3IAPs can be solved using branch and bound techniques, and many have been proposed [139] [22] [94] . MIAPs can be extended beyond three indices, but as can be expected the results become more difficult to find and the approximations become less accurate. One of the best approximations for any general MIAP is a 1/k approximation given by [63] and a 2/k approximation for zero or one cost coefficients given by [70] for the maximization problem.

Hueristic methods for solving NAP are many of the same techniques as used for QAP. GRASP and tabu search as well as simulated annealing and genetic algorithm are used.

2.**3.4 Robust Optimization**

Another key issue in optimization is robustness. Traditionally in optimization, robustness refers to some unknown or uncertain parameter within the function being optimized, or within the set variables which implicitly affect the value of the optimized function. The goal of robustness is to consider the uncertainty in the problem in a certain way as to minimize the effect of that randomness, or account for the possible effects of that randomness. Sometimes a more robust solution with randomness will be worse than the equivalent optimum without randomness, but the addition of that randomness destroys the second case. These issues may be considered along all areas of optimization, such as functional optimization. Stochastic optimization and combinatorial problems are of key interest in business and management problems, or any area in which decisions have to be made under uncertainty and the goal is to minimize the effects of the uncertainty. Laguna [88] considered a problem of project funding, in which a set of n projects is further divided into m categories. Each project has an associated probability of success, and associated cost, an associated profit and an associated variance. The system is also under various structural and control constraints which must be satisfied. The goal of the optimization is to maximize the mean-variance of the expected profit or return. The method of solving this problem was that of the aforementioned GRASP procedure. Laguna found that for a larger portfolio and larger variances within each selection the amount of iterations required increased. It is important to note that this is not a traditional assignment problem. Rather than find pairs for a number of open slots and future projects. However interactions between different categories are linked via constraints such as total cost.

A different example of robust combinatorial optimization conducted by Feige et al [48] also occurred in the area of business management. In this work the specific type of problem is called the two-stage optimization problem. In the two-stage optimization problem, the first stage consists of decisions on what resources should be purchased given only information about the distribution of each resource. In the second stage, exact knowledge of the data is known and the solution is allowed to be complemented by purchasing extra resources at an inflated cost. In the two stage robust optimization problem, a set of possible scenarios replaces the distributions from the non robust problem. In this case the problem is a matter of having an unknown amount of clients, and a set of resources needed to serve those clients. In this case, the set of scenarios is given by an upper bound on the maximum set of clients, rather than a list as has been used in other work [41] . The specific problems focused on in this work deal with issues of covering, specifically the max-min set cover. That is to say finding a subset of k clients whose minimum cost of covering is maximized. Covering in this sense means fulfilling their needs, appeasing them etc. Covering algorithms were developed for this problem in un-weighted instances (i.e. giving all sets unit cost). For this case, the minimum approximation factor is no better than two, and that is the best among other similar problems studied.

Yet another case of dealing with uncertainty in combinatorial optimization is that of the PhD thesis of Morrison [99] . The main idea of Morrison was to treat the persistence of decisions as evidence of robustness. An example of this is to look at a set of acceptable solutions and find commonalities among them, which should suggest that specific decision is robust to uncertainties in the model. The idea behind this is that probability itself need not be the underlying measure of uncertainty. The initial problem is to generate a set of acceptable solutions with good objective values. Next, results are tabulated concerning which individual decisions occurred most frequently and least frequently. Combining this information with the Dempster-Shafer [132] theory of evidence, Morrison was able to create a technique using persistence as a means of evidence of robustness. An example problem of sensor placement was formulated. In essence, this technique is a different type of optimization concerned with expert opinion rather than traditional techniques. Given a set of possible solutions from some source (an expert or someone else), the task is to take that information and parse it in such a way as to find the best and most robust solution, which will likely not be any specific solution given.

CHAPTER 3: RESEARCH METHODOLOGY AND INTEGRATED MODELING AND SIMULATION DEVELOPMENT

This chapter addresses the overall flow of the research discussed in this desecration, and then focuses on the modeling and simulation environment and its development. The main objective of the research is to answer the research questions given above in chapter 1 and reproduced below.

1. Can a metric or set of metrics be found which accurately represent the dynamic effects and changes of the system as it pertains to Cooperative Control? Can these metric(s) be used as a means to trigger changes in the control scheme?
2. Can a control scheme be developed which will consistently perform no worse than the baseline scheme of no communication?

The first step towards this goal is to create a simulation environment which can execute the types of system changes discussed above, implement the desired control schemes and measure their performance. The source code used to create this environment is given in Appendix C. A subset of this environment will ignore control schemes and communication practices in order to study the network itself. This smaller environment will be used to study question 1. The overall environment will serve to create a reduced Monte Carlo in order to study a wide array of network types, and system changes in order to adequately test the control scheme for a wide range of communication networks. If the control scheme is to succeed regardless of what the communication network is, the scheme must be proven to work in a variety of network types and situations. This requires a large amount of different system settings to adequately ensure that these types of systems have occurred and are accounted for. Due to the nature of the high level control analyzed in this research, a test bed is a necessity to analyze its performance. By tracking the performance of individual cases and analyzing if and when failures occur, problematic cases can be studied in more detail until solutions are found. The final control scheme will be evaluated over the entire set of cases for comparative performance reasons and to ensure that the research objectives have been reached. The basic flow of the modeling and simulation environment is given below in Figure 31. The green sections highlighted represent the pieces of the environment used for research question one, while the overall system is used for research question two. Not shown is the visualization section of the environment; this is mainly due to it not being entirely needed for the system to operate. It will still be discussed in this chapter.

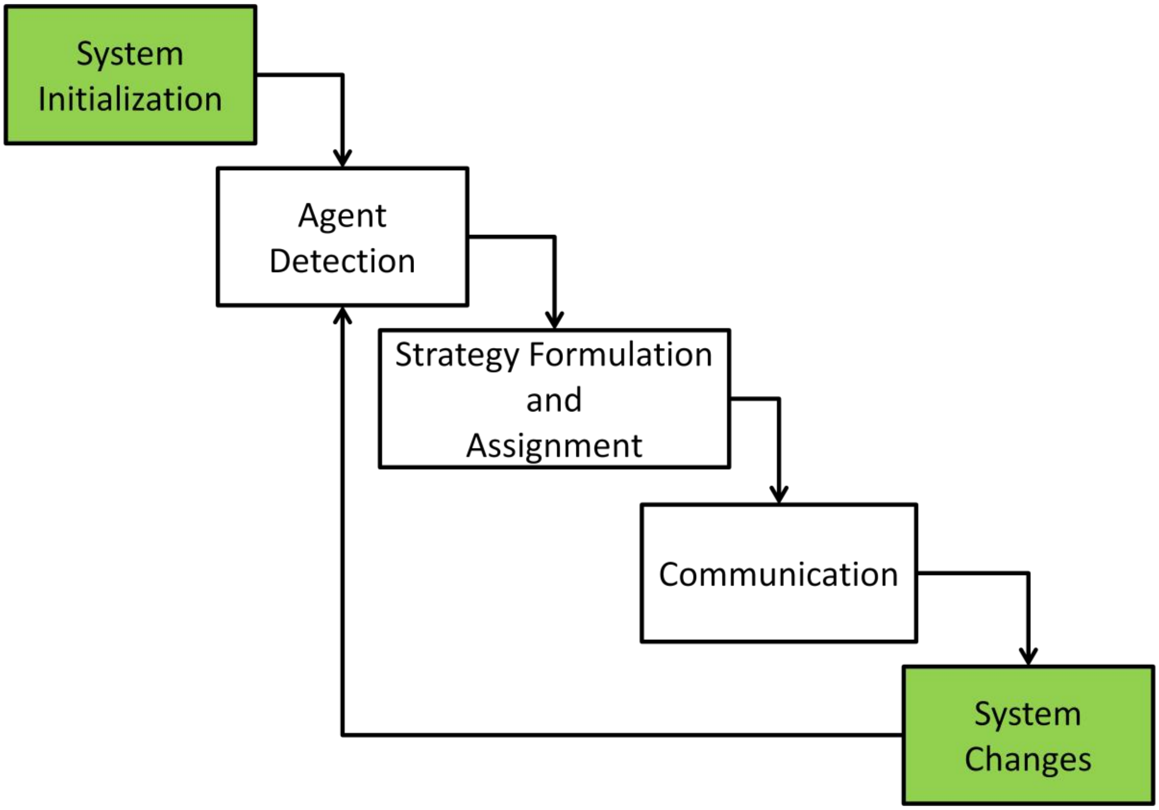


Figure 31 System Flow

3.1 Overview of Environment and its Major Components

The modeling environment contains the following overall components:

- Initialization
- Information Storage
- Overall Metric Determination

- Local Network and Metric Determination
- Strategy and Assignment
- Performance Evaluation
- System Changes
- Visualization

These components will form the major subjections of this chapter. First covered though will be the important variables tracked throughout the system and what they contribute to the system. The only other component discussed in a different order than what is given above is that of initialization and system changes. The reason for this is that those two pieces make up the metric evaluation system, and for that system the control and communication pieces are not needed. With that exception, the other pieces listed above will be followed in order. The overall objective of each section is to describe the goals and basic ideas of each component, as well as the major inputs and outputs. Different options that can be modified will be listed as well as the inputs and outputs of each section. A complete list of all inputs to the environment (and the corresponding ranges of each) can be found in Appendix A. The source code itself can be found in Appendix C.

3.2 Major Variables, Tracking and Other Issues

Before going into detail of the individual sections of the modeling and simulation environment, a few issues must be discussed. The biggest issue of the environment is the allotment of knowledge in the system and the parsing of information each individual actor knows. It is easy to keep track of overall system information but the nature of the problem means that each individual agent will not know or have access to that information in its most up to date form. Given the communication schemes discussed in previous sections, if some agents have knowledge about other agents it will be at some point in the past. The creation and maintenance of each agent's system knowledge must be handled accordingly. In addition to this, some special consideration also needs to be made into the tracking of agents and tasks throughout the system. Because tasks and agents are removed and added at various points in the simulation, some consideration needs to be made to make sure that is handled properly. This is mainly done by an over-all storage variable to have a count of all tasks and agents even those removed. Beyond these issues some of the more important system variables will be discussed and how they are used in a general sense within the environment. Those variables are given below:

- Adjacency Matrix – This represents the current state of the communication structure of the agents within the system. This is simply to show what agents are linked to what other agents.

- Task Matrix – This represents the current state of the task/agent system and what combinations are allowed via assignment. When an agent is linked to a task in this framework, an assignment can be made. Otherwise the corresponding cost for this match is represented by a zero.
- Cost Matrix – This represents the effectiveness of given agent/task pairings to be used in the assignment scheme.
- Metrics – For the control system to be changed based on network properties, it is necessary to determine those properties.
- Knowledge History – This set of metrics is used in order to ensure that each agent's knowledge is based on the proper timed information from its allies. This is the piece necessary for the reasons given above.
- Local Information – As discussed above, each individual agent has its own information about what is going on in the system, but does not have access to the overall current information state. Instead it must build its own idea of each of the above variables and act upon those, not knowing the proximity of this information to the actual current information.

3.3 Initialization and Major System Changes

3.3.1 Initialization

The overall goal of this component of the environment is to create the initial conditions; which include adjacency matrix, cost matrix and task matrix. In the initial state of the system, no knowledge is known of other agents in the system, so those variables will be empty at that time.

Major Inputs:

- Initial number of agents
- Initial number of tasks
- Communication network type
- Additional network properties (1-2 based on network type)
- Initial compatibility parameter between agents and tasks
    - This corresponds to what number or percentage of agent/task assignments are incompatible or have a cost of zero
- Cost matrix properties
    - Minimum cost
    - Maximum cost

Major Outputs:

- Adjacency matrix
- Agent/Task compatibility matrix
- Cost matrix

The first two inputs are clear and merely indicate the size of the initial system for agents and tasks. The network type has four possible options given below:

- Erdos-Renyi Random Graph
- Watts-Strogatz Small World Graph
- Exponential Graph
- Barabasi-Albert Scale-Free Graph

Each of these networks was discussed in Chapter 2 and combined they represent four of the major types of networks found in nature. These were chosen to give a good representation of the popular types of graphs found in the field and each one has special properties previously discussed. Each of these networks has one associated special property which can be scaled, except for the Watts-Strogatz graph which has two. Each of these properties is represented in the system as a percentage between zero and one, but each has a different meaning. For the Erdos-Renyi graph, the percentage represents the percentage for each possible pair wise link to exist. For example, a value of zero means a completely disconnect graph and a value of one represents a completely connected graph. The first variable for the Watts-Strogatz graph represents the number of nearest neighbors each node is linked to. This variable should be a multiple of two. The percentage is multiplied by the number agents in the system and rounded to the nearest multiple of two. The second property variable represents the chance for re-wiring of a given link. Each existing link from the initial linking of neighbors is given a given chance to be re-wired. If it is rewired, that link will be removed and a new link will be added between two agents chosen at random. The final two graphs extra parameter have the same meaning but different execution. For each case the percentage given is multiplied by the total number of agents minus one, to represent the possible number of agents it can be linked to. For the exponential graph, each time a new agent is added, it will be linked at random to that number of existing agents (or as many as possible). In this way agents will be added and linked until the total number of agents desired has been reached. For the Barabasi-Albert scale free network the links are not chosen at random among the already existing agents, but are more chosen with preference. New agents are more likely to be linked to agents which have a larger number of neighbors than agents with a smaller amount of neighbors. With this information the initial adjacency matrix is created fully.

Next is the creation of the agent/task compatibility matrix. This matrix represents which possible assignments can be made and which cannot, and each element is given by a one where assignment is possible and zero where it is not. The desired inputs of this matrix are the minimum and maximum amount of possible assignments that can be made per task. Note that the minimum value must be at least one for each task or the problem will be ill posed. For each task a random value is chosen between the minimum and maximum number of possible agents and then that number of agents at random without preference. These agents can be assigned to the task. With this the Agent/Task compatibility matrix is completed.

The final step is creating the initial cost matrix representing the ability or performance of the possible agent/task assignments. Given a minimum and maximum cost (the cost is set to be between one and zero); each possible assignment as given in the agent/task compatibility matrix is given a random value between the max and min.

With these each of the outputs is given and the initialization component is complete.

3.3.2 Modification of Communication Network (Link Modification)

The goal of this component of the environment is to apply changes to the adjacency matrix via connections between agents. Both addition of new links and removal of existing links will be implemented. One thing to note before going forward is the balance between addition and removal of links. During actual testing, many of the variables given below will be set to equal values to keep the system somewhat stable and not directly influence the system into a disconnected or completely connected network. However for testing and other purposes it might be desired to have those values be different and therefore are allowed to be different. With that said, the major inputs and outputs are given below.

Major Inputs:

- Adjacency matrix
- Change Interval
- Probability of removal of links
- Minimum amount to remove
- Maximum amount to remove
- Amount of removal option
- Method of removal (choosing which to remove)
- Probability of amount to add
- Minimum amount to add
- Maximum amount to add
- Amount of addition option
- Method of addition (choosing which to add)

Major Outputs:

- Updated adjacency matrix

The initial adjacency matrix is needed as it represents the baseline from which changes are made. The change interval effects how often network modification actually happens. If the interval is one, there is a possibility of changes occurring is every single time-step. If the value is five, it may occur every fifth time step. This will be of great importance in the control of the system as more frequent changes have greater impact than those that happen less often. Changes do not necessarily occur each time this interval occurs; the chance of changes occurring is based on the probability of link removal and addition. This feature was included such that the system changes would not always happen when the interval was up but could be made to always happen if desired and give some variability in testing. Next are the minimum and maximum amount to change (add or remove) and what option is used to determine how much is actually changed. The option is one of three choices: the amount to be changed is a fixed amount, such as three to five links; the amount to be a percentage of the current amount of links or the amount to be a percentage of the maximum amount of links. The second option is slightly different for addition and removal options. For addition, the option for the current amount of links represents the amount of open combinations or agents that are not connected. For removal the amount of current links is the number of actual links that exist in the system. This feature simply dictates the number of links that will be added and removed, not which ones are chosen. That feature is dictated by the method of addition and removal which both have three options. The first is a simple choice at random with no preference. The next two have preferences based on the numbers of neighbors each agent has of the two agents the link connects. In one case, addition or removal happens more likely to those links which connect highly connected agents, while in the other addition or removal happens more likely to those links which connect sparsely connected agents. These latter two options have the effect of making the system more or less evenly distributed in terms of the number of neighbors each node has. Another way of saying this is that in one case a few agents will have the majority of the links in the system while many have very few. In anther the distribution of links will be more even.

This covers means of communication network modification without addition or removal of agents in the system.

3.3.3 Cost Changes

The goal of this component of the environment is to change the cost matrix and therefore the performance of individual agent/task assignments within the system. It is important to note that it is not the goal of this component to change compatibility between agents and tasks, and that ability is left to a different component. This is to say that this section will not reduce any costs to zero that were non-zero or make any zero costs into non-zero values.

Major Inputs:

- Cost matrix
- Change interval
- Chance of change
- Minimum cost
- Maximum cost
- Minimum amount to change (cost value)

- Maximum amount to change (cost value)
- Method of change
- Minimum amount to change (elements in the matrix)
- Maximum amount to change (elements in the matrix)
- Method of change

Major Outputs:

- Updated cost matrix

The original cost matrix represents the foundation from which changes are made. Like the above case, the change interval essentially sets the amount of time between possible changes. In the same vein is the chance of change occurring. From here is the minimum and maximum cost. These serve as a bracket for any future changes to occur. No non-zero cost can be below the minimum cost, and no cost can be above the maximum cost. The next variables effect how changes to a single element of the cost matrix once such a change is deemed to occur. There are two methods of change for this problem, a set amount of change and a percentage change from the current value. The maximum and minimum amounts of change correspond to this kind of change.

The next variable set pertains to the amount of elements in the cost matrix itself which will be modified. The two options of change are either choosing a fixed amount of elements regardless of the cost matrix dimension, or choosing a percentage of the elements in the cost matrix. For this case the percentage is applied to the number of non-zero elements within the cost matrix, because the zero values are not allowed to be changed in this section as said above. From here the minimum and maximum amount of change is clear as being simply how much change is allowed.

This covers the possible changes to the cost matrix.

3.3.4 Addition and Removal of Tasks

The goal of this component of the environment is to add or remove tasks to the system and make changes to the overall agent/task compatibility matrix. This section is more complicated than those before it because changes in this component must also make associated changes to the cost matrix. This happens because new compatibilities or incompatibilities may be introduced as well as completely new tasks which will also require changes to the cost matrix. Below are the major inputs and outputs into this section.

Major Inputs:

- Agent/task compatibility matrix
- Cost matrix
- Change interval
- Add chance
- Remove chance

- Minimum added
- Maximum added
- Minimum removed
- Maximum removed
- Total minimum number of tasks
- Total maximum number of tasks
- Link minimum
- Link maximum
- Link addition method
- Minimum cost
- Maximum cost

Major Outputs:

- Updated cost matrix
- Updated agent/task compatibility matrix

The original agent/task compatibility matrix and cost matrix will set the baseline from which changes are made. The change interval represents the amount of time steps between possible changes being implemented, and the addition and removal chance represent the likelihood of these changes occurring. When changes do occur, the number of tasks added or removed is dictated by the minimum and maximum added and minimum and maximum removed factors. As discussed above, the probability and number of tasks added or removed should be kept equal to keep the system stable and not drive it toward the bounds of the system. However this might not always be the desired case so it is left as an option. For removed tasks those chosen to be removed are at random and with no preference. The total number of tasks in the system is bound below with the total minimum number of tasks and above with the total maximum number of tasks.

The next part of this component deals with linking agents and tasks and making them compatible for assignment. When new tasks are added they must be made compatible with some number of agents (at minimum one). This is dictated by the minimum and maximum link variables. When a new task is created, a random number between those two is selected and the task is linked to that number of agents. The selection of which agents to link to is dictated by the link addition method. Three options are available for this, the first being choice at random with no preference. The second and third options have a preference toward linking with agents with a large number of linked tasks or a small number of linked tasks respectively. This was done mainly to look at cases of a good spread between compatibilities among agents, vs. cases when a few agents had much compatibility and others had relatively few amount of compatible tasks. These also come in to play when a task is brought below the minimum number of links due to changes in the system. This can happen due to removal or addition of agents. When a task is found to have less than the minimum number of compatible agents, it is treated as a new task and a random value of new agents are chosen such that its total number of linked agents is between the minimum and maximum amount. Also in this case, the method of selection of which new agents to link to is dictated as above.

Finally in this component the cost matrix must change accordingly. This is due to compatibilities being changed, and new tasks being added or old tasks being removed. Once the new agent/task compatibility matrix is created, the cost matrix is updated such that the corresponding removed task columns are removed, and new task columns are added (with all zero values). From here a check is made to ensure each zero value in the compatibility matrix corresponds to a zero in the cost matrix, or the cost element is changed to zero accordingly. Finally, a check is made to ensure that each one value in the compatibility matrix corresponds to a nonzero value in the cost matrix. For those that aren't, a new proper cost element must be updated. For this reason, the minimum and maximum cost values are included in this component of the environment.

This covers the possible changes to the agent/task compatibility matrix and the other associated changes.

3.3.5 Addition and Removal of Agents

The goal of this component of the environment is to add or remove agents to the system and to make corresponding changes as a result. This is probably the most complex or involved change within the system due to the fact changes in the number of agents has in impact on all other aspects covered above including the cost matrix and the agent/task compatibility matrix as well as the adjacency matrix. Below are the major inputs and outputs for this component.

Major Inputs:

- Adjacency matrix
- Agent/task compatibility matrix
- Cost matrix
- Change interval
- Add chance
- Remove chance
- Minimum added
- Maximum added
- Minimum removed
- Maximum removed
- Removed option
- Minimum total agents
- Maximum total agents
- Link minimum (to other agents)
- Link maximum (to other agents)
- Link options (number and method)
- Minimum cost

- Maximum cost

Major Outputs:

- Updated adjacency matrix
- Updated cost matrix
- Updated agent/task compatibility matrix

The original adjacency matrix, agent/task compatibility matrix and cost matrix represent the baseline from which changes are made. The change interval represents a means to dictate how often possible changes are made, with the addition and removal chances representing the likelihood of those change occurring. The amount added or removed is bounded by the minimum and maximum addition and minimum and maximum removal respectively. Once again, for general use it is important to balance the impact of addition and removal for, but it may be beneficial to have them be unbounded for testing. Added agents have no special means of being added, but removed agents can be chosen in a number of different ways. This is dictated by the removed option variable, and three options are available. The first removal option is that the removed agents are chosen at random and without and preference between agents. The second and third removal options have preference for the agents removed based on the number of neighbors the agents have. The second option gives a higher preference for removing agents with a larger number of neighbors while the third option gives a preference for removing agents with a smaller number of neighbors. This gives the option of skewing the system in terms of the amount of neighbors each agent has, whether it be toward a balanced mix or toward a few agents which have a larger amount of neighbors with the rest having a relatively small amount.

When new agents are added, more options must be considered concerning how they are linked with existing agents and how they are linked with the tasks in terms of compatibility. In order to link new agents to existing agents, the link minimum, link maximum, and link option variables are used. For the specific number, the link minimum and link maximum and link option (number) are used. The option has two choices, the first being a set amount to be added and the second being a percentage of the total possible links to be added. Based on which option is chosen, the minimum and maximum are chosen accordingly. For the linking of agents to tasks, a number is chosen at random between one and the total number of tasks, and those tasks are linked to at random. More methods to effect different effects for compatibility are given in the agent/task compatibility matrix change section above. It is desired that each agent be compatible with at least one task initially and up to as many as are possible. If the compatibility matrix is too sparse, this limits the number of choices each agent can make which also limits the control. In the extreme case, the choice of assignment is singular meaning that regardless of the control, the given assignment must be chosen. This effect will be discussed more in later chapters.

With the additions and removal of agents taken care of, the corresponding agent/task compatibility matrix and cost matrix must be changed accordingly. As discussed in the above section, each of the above matrices will be analyzed vs. the changes made to make sure there are no inconsistencies. When inconsistencies are found, appropriate changes must be made. For this reason new cost elements must be created when new agents are added and the minimum and maximum cost values are needed for this.

This covers the possible changes to the system due to the addition and removal of agents and the other associated changes which must be made to the adjacency matrix, agent/task compatibility matrix and cost matrix.

These system changes combined with the system initialization make up the portion in the system which is beyond the control of any cooperative control technique. It is up to the cooperative control scheme to alter the system based on these changes regardless of what changes are made. This component makes up the evaluation piece for testing different network properties and metrics. The remaining portions of the environment deal with communication amongst agents, assembling the proper knowledge each agent should have from this communication, determination of network properties, implementation of system control, and evaluation of system performance. Finally after these components have been discussed, the visualization aspect of the environment will be covered.

3.4 Information Storage

The purpose of this section is to store information from each time step for later use in the simulation. The reason for this is because in the creation of the knowledge for each agent, some of that information will be old and from sources which may or may not currently exist in the simulation. For this reason it is convenient to store the information available in the system in one place for each time step and information will be selectively taken to form the knowledge of each agent. Below are the major inputs and major outputs for this component of the environment.

Major Inputs:

- Current adjacency matrix
- Current cost matrix
- Current agent/task compatibility matrix

Major Outputs:

- Information storage at each time step including:
    - Adjacency matrix
    - Cost matrix
    - Agent/Task compatibility matrix

The information stored in this component is not directly used immediately, but will be used in later sections.

3.5 Overall Metric Determination

The purpose of this section is to determine and calculate key system metrics for the communication network as changes occur within the system. This is mainly used for direct measure of the system in finding key parameters to represent the system, but may also be used for testing and debugging. Each of the metrics calculated will be given in its raw state, and normalized by the number of agents in the system and the total number of links in the system. In addition to these static metrics, dynamic metrics will also be calculated simply by computing the difference between the current metric value and the previous one. The choice of metrics will be discussed in the next chapter. The major inputs and outputs of the system are given below.

Major Inputs:

- Adjacency matrix

Major Outputs:

- Diameter
- Connectivity (2)

- Cheeger constant (2)
- Number of links in the system
    - Each of these static values is normalized in the above mentioned ways
    - Each of these static values are also stored in order to capture their changes over time

From here a brief explanation of each metric and what it defines about the system will be given, as well as how each is calculated. The diameter of the system represents the worst case scenario for how long it will take information to travel across the system. This is of key importance to the control because it dictates how long it will take after a major system change for the rest of the system to receive that information in a conservative sense. In general, diameter is only defined for a system that is connected, however for comparison purposes the diameter of a disconnected network will be defined as the size of the system squared. This is mainly just to provide an easily seen difference in the system when it is disconnected. The diameter is calculated in the following way, first take the adjacency matrix and create a new dummy matrix of the same size with all zero values. For each element of the adjacency matrix that is non-zero, place a one in the corresponding dummy matrix. As long as at least one element in the dummy matrix is zero, the process will continue. From here, multiply the adjacency matrix by itself and repeat the process of mapping the non-zero elements of this matrix to the dummy matrix, however only overwrite zero values in the dummy matrix. Once the dummy matrix has no zero values, the number of iterations represents the diameter. This process is given below in Figure 32 where A represents the adjacency matrix of the system. It is important to note that the maximum diameter for a connected system is given by the number of elements in that system minus one. If the loop continues on beyond this process, the network is disconnected.

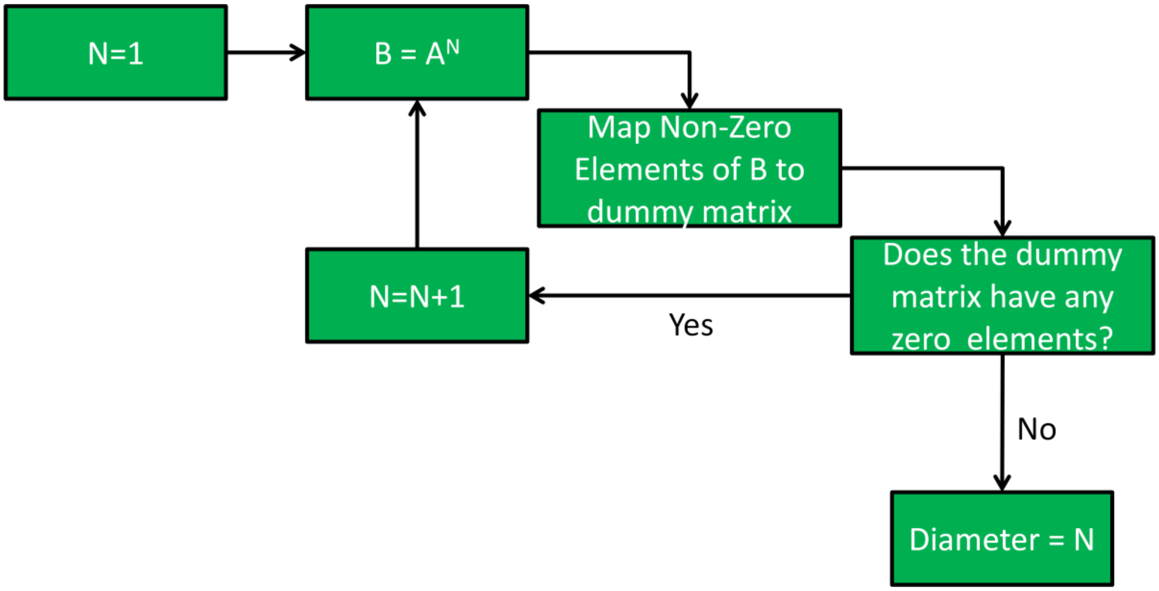

N=1
B = A^N
Map Non-Zero Elements of B to dummy matrix
Does the dummy matrix have any zero elements?
Yes
N=N+1
No
Diameter = N

Figure 32 Diameter Determination Process

The reason why this process works is because raising the adjacency matrix to a given power (N) has a special physical meaning to that network. Each non-diagonal element (i,j) represents the number of paths between the two corresponding nodes (i,j) of length N. By systematically searching for all paths in increasing length, eventually it will be found that all path exists and the largest will be the diameter, or that the system is disconnected and the diameter has no traditional meaning.

The next metric calculated is the connectivity or sometimes called the algebraic connectivity. In general, the connectivity has two different meanings, the first is based on Eigen-value analysis and the second is a ratio of the number of connected triples in the system to the possible number of connected triples in the system. Both can be thought of to represent the health of the network in terms of the links and how well it is connected. For both measures, the smaller the connectivity the more sparse its links and conversely the larger the connectivity the more densely connected it is. A lower connectivity can be thought of as that network being more unstable and more likely to become disconnected by removing links within the system. To calculate the first connectivity, the adjacency matrix must be converted into the Laplacian matrix (which is defined in Chapter 2). From here, the Eigen-values are taken of this matrix with the connectivity taken as the second smallest Eigen-value. If this value is zero, the system is disconnected, which can be used to more easily determine the diameter. This connectivity value ranges from zero to the number of elements in the network. A connectivity of the highest value would correspond to a diameter of one in the system or conversely that the network is fully connected. For the second connectivity, the ratio is calculated using Equation 31 below where A is the adjacency matrix and O is a matrix of ones of corresponding size.

Equation 31 Connectivity

$$Connectivity = \frac{trace(A^3)}{trace(O^3)}$$

The reason why this equation works is because the trace of the adjacency matrix cubed represents all connected triads, and the denominator exists to represent all triads in a fully connected graph (or the maximum value). Unlike the previous connectivity, this value ranges from zero to one; a value of one also implies that the system is fully connected and has a diameter of one.

The next metric is the Cheeger constant. The precise mathematical definition is given above in chapter 2. This is another measure of the stability of the network, like the connectivity. In this case, this metric represents the “bottle-neckedness” of the system or conversely how easy it is to separate the system in the separate disconnected components. A smaller number represents a system more easily separated and a larger number represents a more stable and highly connected system. However, calculating this value is not easy and there is no computational method which can ensure its calculation in a given number of iterations [7] . For this reason it is easier to use bounds on the Cheeger constant instead of its actual value. The bounds of the Cheeger constant are based on the second Eigen-value of the normalized Laplacian matrix where element i,j is normalized by the square root of element i,i times element j,j of the un-normalized Laplacian matrix. Those bounds are given below in Equation 32.

Equation 32 Bounds on Cheeger Constant

$$\frac{e_2}{2} < Cheeger\ Constant < \sqrt{2 * e_2}$$

The final metric is also the easiest to understand and calculate. It is simply the number of links in the current network. This is calculated below by Equation 33, where the two summations are over the rows and columns of the resulting matrix.

Equation 33 Number of Links in the System

$$\sum\sum\frac{(A - I_n)}{2}$$

This simply adds the number of non-diagonal elements and divides by two to prevent double counting. This is perhaps the most brute-force way to calculate the stability of the network because it gives no relative weight to the links in the system, simply that they exist. On this same note, this can be thought of as an upper bound of sorts to the connectivity once normalized.

Finally a brief statement on the normalization that occurs for these metrics will be given. Because many of these metrics are directly related to the size of the system, to compare different network sizes it is important to normalize them. Only two real options make sense. The first is to normalize by the size of the network, or the number of agents within it. The second is to normalize by the total number of possible links in the system, which is given below in Equation 34. This is a more suitable quadratic normalization for the network.

Equation 34 Total Possible Links in a Network

$$\frac{(N)(N-1)}{2}$$

This finalizes the metrics calculated within the system.

3.6 Local Network, Cost and Metric Determination

The purpose of this section of the environment is to determine the local structure of the network from the point of view of each agent in that network. The overall goal of the system is to change its behavior between aggressive and conservative based on the properties of the network in order to best adapt to the changes in the system. In order to achieve this goal, the local knowledge must be determined because there is no way for each individual agent to know the current accurate state of the overall system, only what has been communicated to it by other agents. The difficulty of the problem arises from this fact, because when changes occur to the system, they will not be known immediately to all remaining agents not immediately impacted by that change. In some cases it may take a considerable amount of time for that change to be known. Another result of the system is that any knowledge gained from other agents will be older than the current state of the system. These factors must be adequately modeled and account for. The output of this section is that each agent will have an idea the overall network structure. From here, the local network can be used to formulate local metrics, and the local cost will be used for assignment.

Major Inputs:

- Adjacency matrix
- Cost matrix
- Stored cost matrix
- Stored adjacency matrix
- Neighbor knowledge matrix

Major Outputs:

- Updated neighbor knowledge matrix
- Local adjacency matrix
- Local cost matrix

- Local metrics
    - Diameter
    - Connectivity (2)
    - Cheeger constant (2)
        - Each of these static values is normalized in the above mentioned ways
        - Each of these static values are also stored in order to capture their changes over time

The current and stored versions of the cost and adjacency matrix are used as a baseline from which pieces are taken to form the local data for each agent. The crux of this portion of the code and ensuring it works properly is closely related to the neighbor knowledge. This variable is represented by a matrix the same size as the adjacency matrix. Each row represents the knowledge of that agent for each other agent in that corresponding column. In other words, for element i,j a value of four means that agent i has knowledge of agent j's data which corresponds to time step four. Given below in Table 37 and Table 38 are example neighbor knowledge variables for a fully connected communication network and an empty or isolated communication network respectively (for time step five). It is important to note that no matter the structure of the network, the diagonal of this matrix will be the current time step for each element. All other elements must be at least one less than the diagonal elements due to the fact that all other knowledge is at least one time step older than the current information. Of final importance is that a value of zero in this matrix indicates that agent i has no knowledge of agent j or any of its information. Another way to say this is that agent i does not know that agent j exists. For this simulation, it may be better to assume no knowledge rather than using excessively old knowledge as the older knowledge becomes the more likely it is no longer relevant and may do more harm than good.

Table 37 Example Neighbor Knowledge of Fully Connected Network

5	4	4	4	4	4	4	4
4	5	4	4	4	4	4	4
4	4	5	4	4	4	4	4
4	4	4	5	4	4	4	4
4	4	4	4	5	4	4	4
4	4	4	4	4	5	4	4
4	4	4	4	4	4	5	4
4	4	4	4	4	4	4	5

Table 38 Example Neighbor Knowledge of Isolated Network

5	0	0	0	0	0	0	0
0	5	0	0	0	0	0	0
0	0	5	0	0	0	0	0
0	0	0	5	0	0	0	0
0	0	0	0	5	0	0	0
0	0	0	0	0	5	0	0
0	0	0	0	0	0	5	0
0	0	0	0	0	0	0	5

Next, the creation and updating of this variable will be discussed. To start the simulation, the neighbor knowledge matrix is given by the identity matrix, meaning that the system starts with no knowledge from neighbors. From the first communication, it will be known which agents are immediate neighbors. These neighbors will communicate their own part of the neighbor knowledge matrix (a given row) from which it will be compared to the agents own knowledge. For each column, if the communicated value is higher than the agents own value, the agents own value will be replaced by the communicated value (in a new matrix, to ensure that knowledge is no transmitted early and incorrectly). This is mathematically equivalent to saying if the communicated values are more recent than what is currently known, the communicated values will be used. This will happen for all agents until the step ends. From here, the diagonal values will be increased by one, since those values are the most up to date. Below is a pseudo code example of this procedure.

```
For each agent
    For each neighbor of that agent
        For each column of the neighbor knowledge matrix
            If communicated value is greater than the current value, replace
            the communicated value by the current value
        End
    End
End
```

An example of this procedure will now be discussed. A fixed communication network given by a line network given below in Table 39 and Figure 33 will be used.

Table 39 Example Line Network Adjacency Matrix

$$\begin{vmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{vmatrix}$$

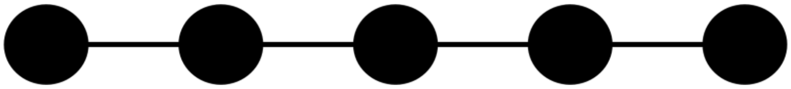


Figure 33 Example Line Network Graphical Representation

The system starts with the identity given below in Table 40.

Table 40 Example Neighbor Knowledge Time Step 1

$$\begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

After one time step each agent will gain the previous time step knowledge of each of its neighbors. This is represented below in Table 41.

Table 41 Example Neighbor Knowledge Time Step 2

$$\begin{vmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{vmatrix}$$

From here the system updates as such until the simulation ends. For the sake of this example, this updated procedure will stop once each agent has knowledge of all other agents. This is given below by Table 42, Table 43 and Table 44.

Table 42 Example Neighbor Knowledge Time Step 3

3	2	1	0	0
2	3	2	1	0
1	2	3	2	1
0	1	2	3	2
0	0	1	2	3

Table 43 Example Neighbor Knowledge Time Step 4

4	3	2	1	0
3	4	3	2	1
2	3	4	3	2
1	2	3	4	3
0	1	2	3	4

Table 44 Example Neighbor Knowledge Time Step 5

5	4	3	2	1
4	5	4	3	2
3	4	5	4	3
2	3	4	5	4
1	2	3	4	5

From Table 44 onward, each value will increase by one per time step unless some major change occurs in the network structure. It is important to note the relation between this matrix and some of the key network metrics studied in previous sections. The diameter of this system is four, and is equal to the largest difference in information age in this system (in this case for the first and last agent, with values of five and one). In this case the idea of a radius is also clear. The third agent is the "central node" of the system and therefore has the most recent information compared to all other agents. In this case the radius is two and that represent the difference in information age for this central node.

One note needs to be made about this procedure concerning changes in the system. Ideally, if no major changes occur, the system will eventually reach some sort of steady state equivalent to that seen in the previous example. In such a case, each value of the matrix will increase by one at every time step. However when a change occurs this steady updating may be halted and some or all of the values may not update. An update may not occur simply because a communication link was broken, or because the communicating agent is no longer active. It will be unknown what the cause of this was immediately after it happens, but may become known at some later time. For reasons which will be discussed more in depth within the control section of this document, the worst case scenario is always considered to have occurred in such an event. When the new information is not more recent than the old, the agent will assume it has no knowledge of that agent (IE that the agent in question has been removed from the system). Another way to say this is that when the updated neighbor knowledge is equal to the previous for a given cell, rather than leave that value the way it is it is replaced by a zero which corresponds to no knowledge of that agent.

Now that the means of constructing the neighbor knowledge matrix are known, how it is used will be discussed. It is mainly used to construct each agents own idea of the communication network as well as each agents idea of the cost matrix of the system. In each case the value of neighbor knowledge matrix represents the time step of information which is available from the particular other agent. Storage of the overall system information for each time step allows an easy access point for combining different values. The creation of the cost matrix is easier to describe in detail as compared to the adjacency matrix and will be discussed first. A set of example cost matrices will be used to demonstrate this creation, each representing a notional cost matrix as it changes in the system. Those are given below in Table 45, Table 46 and Table 47, which are permutations of each other.

Table 45 Local Cost Creation Example Time Step 1

$$\begin{vmatrix} 0.3 & 0.6 & 0.9 \\ 0.6 & 0.9 & 0.3 \\ 0.9 & 0.3 & 0.6 \end{vmatrix}$$

Table 46 Local Cost Creation Example Time Step 2

$$\begin{vmatrix} 0.6 & 0.9 & 0.3 \\ 0.9 & 0.3 & 0.6 \\ 0.3 & 0.6 & 0.9 \end{vmatrix}$$

Table 47 Local Cost Creation Example Time Step 3

$$\begin{vmatrix} 0.9 & 0.3 & 0.6 \\ 0.3 & 0.6 & 0.9 \\ 0.6 & 0.9 & 0.3 \end{vmatrix}$$

The example neighbor knowledge matrix is chosen to represent a line network and is given below in Table 48.

Table 48 Example Neighbor Knowledge Matrix

3	2	1
2	3	2
1	2	3

Below are the local cost matrices constructed from the above information given by Table 49, Table 50 and Table 51.

Table 49 Local Cost Matrix of Agent 1

0.9	0.3	0.6
0.9	0.3	0.6
0.9	0.3	0.6

Table 50 Local Cost Matrix of Agent 2

0.6	0.9	0.3
0.3	0.6	0.9
0.3	0.6	0.9

Table 51 Local Cost Matrix of Agent 3

0.3	0.6	0.9
0.9	0.3	0.6
0.6	0.9	0.3

As can be seen from this example, each cost matrix is incorrect and each different from the other agent's cost matrices. This example was used to demonstrate the differences possible and the problems which may arise due to the communication constraints in terms of information agreement. The only problem which may arise in this procedure occurs when the cost matrices are of different size. In this case, values of zero are used to fill empty values which other values do not exist for. This represents a conservative case of knowing that a new agent or task exists, but not knowing about it completely.

When determining the local network, more problems exist because any non-diagonal value can be gathered from two sources. This idea is demonstrated graphically below in Figure 34. Each agent contributes one row and column (due to the symmetric nature of the matrix), and when combined more disagreements exist.

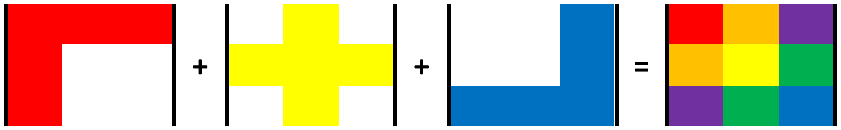


Figure 34 Visual Example of Local Network Creation

For this case, when there is a disagreement the most recent knowledge is chosen if possible. When it is not, the most conservative value is chosen (which will be a zero).

That is to say that when two agents give information which is conflicting over whether or not a certain link exists, the acting agent will assume it does not exist.

Now that the local cost and adjacency matrix are known, their use will be discussed. The local cost is used exclusively in the assignment process in control, and will be discussed in later sections. The local adjacency matrix is used to determine the local metrics of the system. Those are the same metrics discussed in the previous section. This finalizes this section.

3.7 Strategy and Assignment

This section of the environment will discuss the means by which strategy is used and assignment made. Strategy is dictated by the control scheme implemented. This section will not discuss how and why the strategy is chosen, that will be left for the control section of this document. Given that strategy, this is the means it will be used to enact an assignment. Below are the major inputs and outputs to this section.

Major Inputs:

- Local Info
- Strategy
    - Desired redundancy
    - Distribution variable

Major Outputs:

- Assignment
- Ideal Assignment
- No Communication Assignment

The local info includes the previously determined local cost, local adjacency matrix and local metrics. From this the strategy is determined. The strategy is made up of two variables, the first being the desired level of redundancy. This redundancy that the assignment will ensure at least that many agents will be assigned to each task where possible. Based on the structure of the cost matrix, it may only be possible for fewer than the desired redundancy of assignments to happen. In this case the maximum amount of assignments will be made to that task. The next part of the strategy dictates the way in which the assignment is distributed when multiple assignments are made. The maximum amount of assignment "resource" that can be used is one, which means that the row sum of each row in the assignment matrix must be one. When multiple assignments are made a decision must be made about how to divide those resources. To properly optimize considering this fact makes the problem non-linear which is not desired due to its difficulty. In addition, each optimization which occurs is optimizing for a specific instant in time rather than the overall system goal of preventing uncovered tasks. To properly optimize this system would be difficult and doing it a different way can work with some loss of optimality. This was found to be a good way of allowing a good instantaneous solution, while allowing redundancy needed to prevent unassigned tasks. A series of sequential linear assignments (the first assignment is stronger than the next) are made and it is left to afterwards to determine how those resources are split. In order to keep the control scheme simple that means was reduced to one variable. This variable essentially dictates how much one assignment gets in comparison to the one before it. IE in the simplest case the value will be one, and that means each assignment is equal identically to the one before it. An example of other values is given below in Table 52.

Table 52 Examples of Distribution Variable Values

Distribution Variable	**First**	**Second**	**Third**	**Fourth**	**Fifth**
1.0	0.20	0.20	0.20	0.20	0.20
0.9	0.24	0.22	0.20	0.18	0.16
0.5	0.52	0.26	0.13	0.06	0.03
0.2	0.80	0.16	0.03	0.01	*0.00*

The idea behind this is that in most cases the extra redundancy of this system exists solely to protect against the worst case scenarios when dramatic change occurs. Redundancy adds nothing to the optimality of the system except to help ensure against failure (by removing or reducing unassigned tasks), it does not increase the performance of the assignment for one instant in time. As such it is reasonable to assume that the earlier assignments are the main drivers of the performance and should be stronger than the later assignments which are more for redundancy and preventing system failure via unassigned tasks.

These variables feed into and drive the assignment which will now be discussed. There are three assignments which take place, the ideal assignment assuming perfect knowledge of the system at all times, the no communication assignment case (baseline) which is what takes place when each agent has no other information from the other agents in the system and finally the standard assignment which is the goal of this research and takes into account all communication related issues. The easiest of these to determine is the no communication case, as the effective cost matrix is only one row. In this case the agent splits its assignment evenly over every task it can make assignments with. For example, if the cost matrix of the system is given by Table 53 below, the assignment will be given by Table 54.

Table 53 Example Cost Matrix of No Com Assignment

0.10	0.52	0.00	0.87	0.56
0.00	0.06	0.21	0.00	0.29
0.59	0.78	0.33	0.64	0.65

Table 54 Corresponding No Com Assignment

0.25	0.25	0	0.25	0.25
0	0.33	0.33	0	0.33
0.2	0.2	0.2	0.2	0.2

The ideal and actual assignment will be done in the exact same way, with the only difference being the strategy and what cost is used. The idealized case has the benefit of perfect knowledge and will use the bare minimum level of redundancy to ensure that no tasks are unassigned. This case can transition immediately to changes and therefore does not require the added redundancy. In this case the current cost matrix is used. For the actual case, the strategy used will be determined by the control of the system, and the cost used will be the local cost, whose creation is described above. The reduced assignment is essentially given below by Equation 35 with constraints given by Equation 36. Notice that a few of the constraints have been removed or relaxed to facilitate the modified assignment.

Equation 35 Linear Assignment Problem

$$\max\left[\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}\, x_{ij}\right]$$

Equation 36 Associated Linear Assignment Constraints

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,\dots,n)$$

$$x_{ik} \geq 0 \qquad (i,k = 1,2,\dots,n)$$

A few things must be done before assignment in order to ensure proper and desired behavior. The first is pre-conditioning of the cost matrix. There are two main issues that need to be handled, the first being non-unique solutions and the second being assignment of zero cost tasks. For the first problem, if a cost matrix has a non-unique solution it causes problems when some agents will pursue one solution and some do another. This can be solved by adding small values to the cost matrix based on its row and column to the maximum values. For this case, a value of .01*i + .001*j was used. This is a similar idea to giving priorities to certain tasks and agents when such a situation arises. For the second problem, the nature of the cost matrix might give a solution in which an agent is assigned to a task which is incompatible due to the advantage of all the other assignments. This problem is also solved in an easy fashion, by replacing all zero values in the cost matrix by a large negative value, namely -500. This ensures that all proper assignments are made with compatible agent task pairs.

The next task which needs to be done before standard assignment is determining which assignments are pre-determined based on the desired redundancy. If a certain task has greater number of compatibilities than the desired redundancy level, than a decision can be made about which assignments to make. Otherwise the assignments are pre-chosen to attain the desired redundancy or at least as close as possible. This process is called pre assignment. To demonstrate this, an example cost matrix is given below in Table 55 with highlighted incompatibilities.

Table 55 Cost Matrix for Pre-Assignment Example

0.00	0.23	0.00	0.24	0.00
0.00	0.13	0.00	0.50	0.00
0.83	0.63	0.84	0.84	0.00
0.00	0.05	0.40	0.11	0.00
0.49	0.06	0.59	0.82	0.64

If the desired redundancy is three, than tasks (columns) one, three and five must be assigned with the possible agents to be as close as possible to that number. In such cases, the corresponding columns are removed from the cost matrix for the standard assignment process below.

The actual assignment process has two major possibilities, based on whether or not there are more tasks or agents in the system. Mathematically, the easier case occurs when there are an equal or greater number of tasks than agents. In this case the cost matrix does not need to be modified further in order to progress. For the other case, when there are a greater number of agents than tasks, columns of "zero" values are added at the end of the cost matrix until it is square. This is to ensure that only one agent is assigned to each task per standard assignment, and the repeating of this standard assignment takes care of redundancy. The basic concept of the assignment is simple; based on the desired level of redundancy determine the number of individual assignment rotations that are needed. This is given by Equation 37 below.

Equation 37 Number of Rotations in the Assignment Phase

$$Number\ of Rotations = ceiling\left(\frac{tasks * level\ of\ redundancy}{agents}\right)$$

In each case one standard rotation ensures that all agents are assigned (if possible) and all tasks are assigned. For the case where there are more tasks than agents, some agents will need to be assigned multiple times during this rotation. For the case where there are more agents than tasks, some tasks will be assigned multiple times. Doing so will ensure that each rotation is a proper assignment with all possible actors involved. From here the assigned combinations will be “zeroed” out from the cost matrix, and the process will repeat. These values are zeroed out to ensure that future assignments are unique and not repeats of old assignments. This overall process may result in over-redundancy for cases of more agents than tasks but will give at least the desired level of redundancy. As an added precaution an extra check is made at the end of the process for tasks with fewer agents than the desired redundancy.

During one standard rotation, the basic standard linear optimization given below in Equation 38 is solved.

Equation 38 Basic Optimization Problem

$$max(x'f)$$

And are subject to the following constraints given below in Equation 39.

Equation 39 Basic Optimization Constraints

$$Ax \leq B; A_E x = B_E$$

And the matrices f, A, B, A_E, B_E are determined based on the problem and cost. The determination of f can be given below in pseudo code.

```
If #agents>#tasks
    For j=1:#tasks
        f(1+(j-1)*#agents:j*#agents)=cost(j,:)
    End
Else
    For j=1:#allies
        f(1+(j-1)*#tasks:j*#tasks)=cost(j,:)
    End
End
```

Where A, and A_E are given by the pseudo code below.

```
If #agents>#tasks
```

```
        For j=1:#agents
                For k=1:#agents
                        A(j,j+#agents*(k-1))=1
                        AE(j,k+#agents*(j-1))=1
                End
        End
Else
        For j=1:#agents
                For k=1:#tasks
                        A(j,j+#tasks*(k-1))=1
                        AE(j,k+#tasks*(j-1))=1
                End
        End
End
```

The B and B_E matrices are simple column matrices of all ones of corresponding size.

For each case there may be a required amount of sub rotations beyond the initial, unless the number of tasks and agents is equal. For either case, Equation 40 below will work for the number of sub rotation.

Equation 40 Number of Sub Rotations

$$Number\ of\ Sub\ Rotations = ceiling\left(\max\left(\frac{tasks}{agents}, \frac{agents}{tasks}\right)\right) - 1$$

During sub rotations, entire columns are “zeroed” out based on the number of assignments that corresponding task has received. This ensures that each task will receive at least as many assignments as the desired level of redundancy.

3.8 Performance Evaluation

The purpose of this section is to calculate the performance of the studied control. This is done by normalizing performance with the ideal communication case and no communication/baseline case. Below are the major inputs and outputs to the section.

Major Inputs:

- Assignment matrix
- Ideal assignment matrix
- No communication assignment matrix
- Cost matrix
- Previous change history

Major Outputs:

- Number of unassigned tasks

- Frequency of unassigned tasks
- Normalized average task performance
- Normalized minimum task performance
- Possible reasons for system failure

The corresponding assignment matrices serve as the starting off point for comparison. The cost matrix is used to evaluate the initial performance of each, by determining the raw and effective task performance from each task. Finally the history of previous major changes in the system is kept in order to determine what sorts of changes cause failures in the system when they occur.

Before any of the normalized metrics can be calculated, some raw performance needs to be measured. For each assignment matrix, all cost values are multiplied by their corresponding entry in the assignment matrix to give a raw task performance for that pair. In other words, element i,j of the raw task performance matrix is given by Ai,j multiplied by Ci,j where A and C are the corresponding assignment and cost matrices. With this matrix, the effective task performance for each task is found by taking the maximum value in each column. The average of these values creates the average effective task performance and the minimum creates the minimum effective task performance. A zero value in the minimum effective task performance corresponds to a failure in the system due to an uncovered task, and the number of those is calculated and compared with the total number of tasks at each time step in the system. Each of these variables is useful in their raw form but do not tell the entire story, and must be normalized to give some meaning for different networks and cases. The first normalization (total normalization) used is given previously in Equation 1 and is reproduced below.

$$N_T = \frac{A - B}{I - B}$$

Where N is the normalized value, and A, B and I are the actual, baseline and ideal values respectively. Ideally the normalized values should range between zero and one, with values less than zero corresponding to performance worse than the baseline case. The only problem with Equation 1 is the possibility that the ideal and baseline performance are equal, which would make the normalized value undefined. This occurs when only one possible solution exists, and the actual, baseline and ideal performance are equal. In such a case, the normalized performance is defined to be 0.5 which was chosen as the average between baseline and actual performance. When a failure does occur, any system changes in the previous few times steps are recorded, as well as when they happened. This is useful in checking the system and why failures occur when making changes to the control scheme.

The second normalization scheme is given previously in Equation 2 and is reproduced below.

$$N_B = \frac{A}{B}$$

Once again, A is the actual performance and B the baseline performance. This metric can be any non-negative value, with a value of one representing actual performance equal to baseline performance. This value can be thought of as a performance increase vs. the baseline vase. For example, a value of 1.5 would represent a fifty percent performance increase vs. the baseline case. If the problem is well posed, than the baseline performance will always be greater than zero. This fact implies that this normalization has no mathematical issues relating to division by zero. This concludes the performance evaluation section.

3.9 Visualization

The purpose of this section is to describe the visualization process used for the system, what is shown and why it is useful. It may be important to read later sections on some details of the control scheme beforehand because some of those features will be included in the visualization. However it is included in this section of the document as it is part of the environment. The main goal of the visualization is to show many views of the system as it is complicated and difficult to view in one basic graph or chart. The top nine views are included, and will be discussed separately. Below is the overall environment shown in Figure 35.

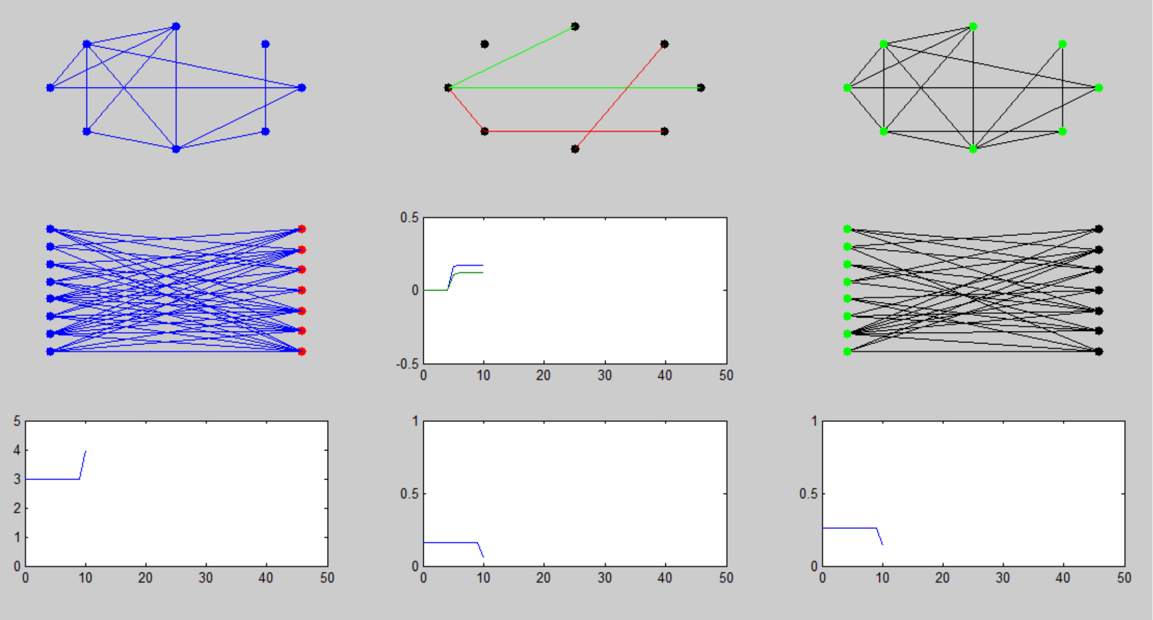


Figure 35 Visualization Environment

Each of the nine charts and graphs have a different meaning or representation and will be discussed starting from the top left, going left to right then down. Below in Figure 37 is the first view. This is a basic view of the communication network as it currently stands. A circle view (the nodes positioned in a circle) was chosen for the graphs because it makes it easier to see all the links in the system. This is mainly just to get an overview of the network in its basic form.

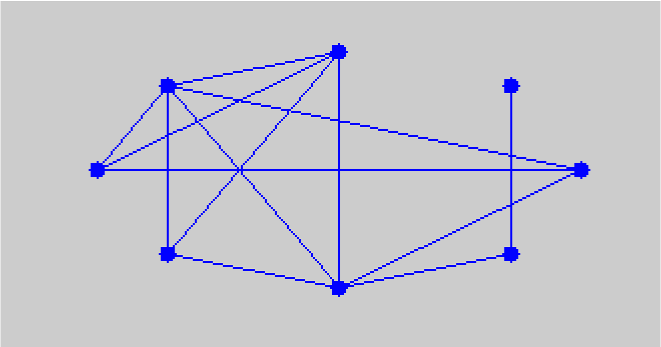


Figure 36 Basic Communication Network

The next view is meant to emphasize changes in the communication network and is given below in Figure 38. This figure shows the nodes of the system and only shows links when they are added or removed. When links are added to the system, they are shown as green and when they are removed they are shown as red. This view greatly emphasizes when changes occur and their impact can then be analyzed in other views.

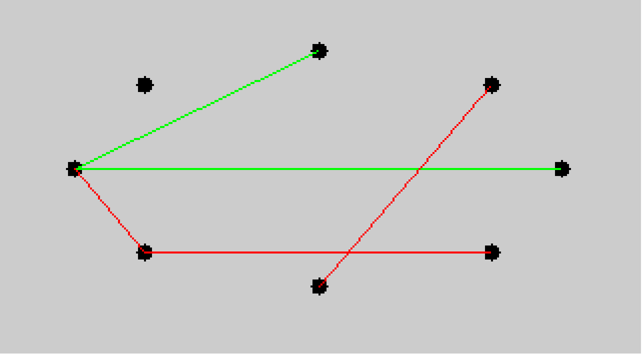


Figure 37 Communication Network Emphasizing Changes

The next view is another view of the basic communication network, but with an added feature in the nodes/agents. This can be seen below in Figure 38. There are two basic states in the control space that an agent can be in based on the properties of the system at the time. Essentially one is when information is in flux and agreement is highly unlikely, while the other is when agreement is possible (called pre-consensus). This view is mainly to show which state each agent is in, and is demonstrated by the color of each node in the network. Black represents the first case of information flux, and green the pre-consensus state.

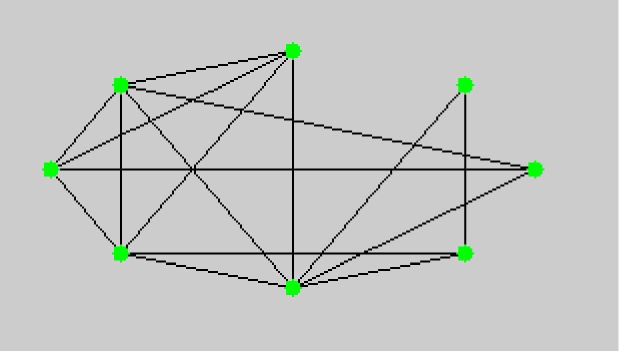


Figure 38 Communication Network Emphasizing Agent Control State

The next view is meant to show the relation between the agents and tasks, and is given below in Figure 39. This view is essentially the compatibility matrix in graphical form. It demonstrates all possible assignments between agents (blue/left) and task (red/right). This is mainly to get an idea of what possibilities exist for assignment and a semi view of how healthy the system is in terms of choices.

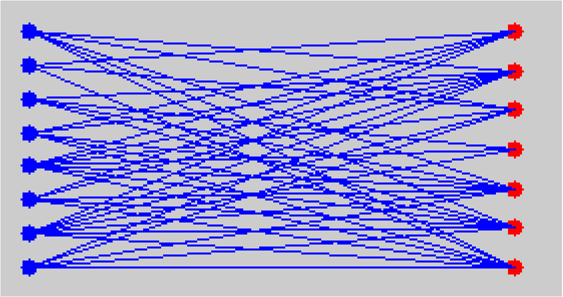


Figure 39 Bipartite Graph of Agent Task Compatibility

The next view shows the overall important performance metrics in the system and is given below in Figure 40. This shows the normalized (total) minimum and average task performance in green/blue respectively. Important issues to note are when values are above or below zero. The first represents cases of total or partial pre-consensus of all agents and the second represents system failures.

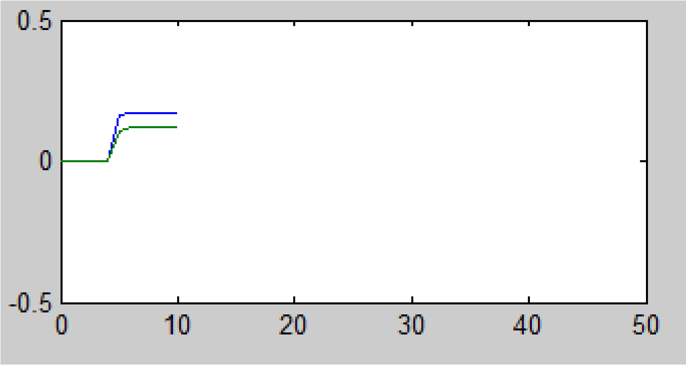


Figure 40 Normalized Minimum and Average Task Performance

The next view represents the assignment of agents and tasks in addition to the state of each agent. This is given below in Figure 41. As in Figure 39, the agents are given on the left and the tasks on the right. Each link is an actual assignment rather than all possible assignments. As in Figure 38, agents are green when in pre-consensus and black when they are not.

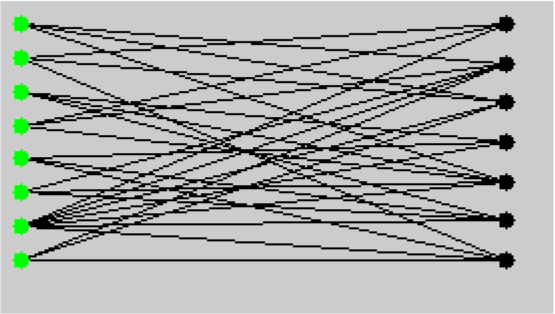


Figure 41 Agent Task Assignment Graph

The final views are of the important metrics in the system, given in Figure 42, Figure 43 and Figure 44. These views give an idea of the stability of the system, where lower values are better for the diameter, and larger values are more stable for the connectivity and Cheeger constant.

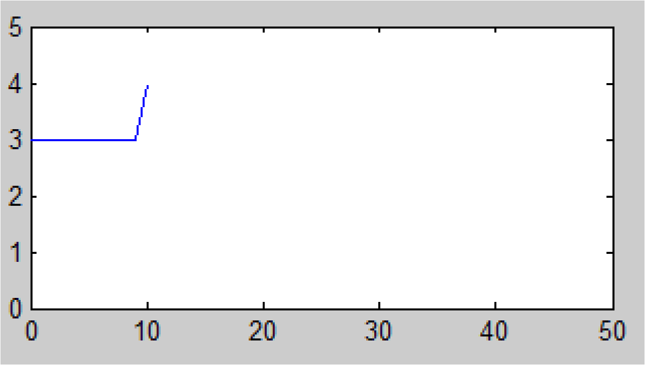


Figure 42 Diameter of Communication Network

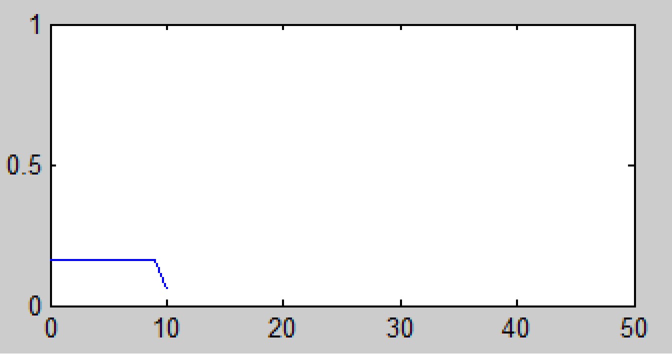


Figure 43 Normalized Connectivity of Communication Network

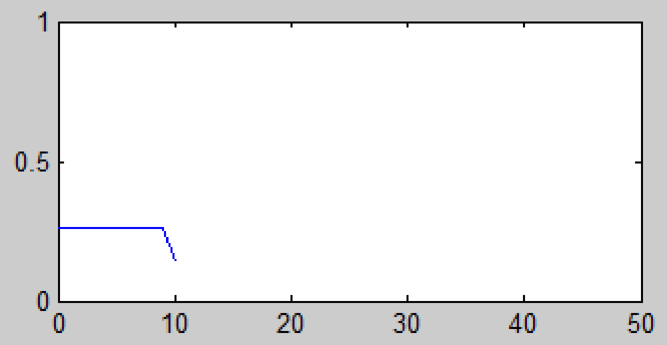


Figure 44 Cheeger Constant of Communication Network

Below are two examples of simulation runs for volatile and a stable building test case to show off the visualization. The first case is a normal volatile run and is represented in Figure 45 and Figure 46. The time history of the metrics of interest shows non-monotonic behavior, as well its frequency of changes both of which are a clear representation of its volatility. Accordingly the overall performance metrics are also volatile, but outperform the baseline case for all times.

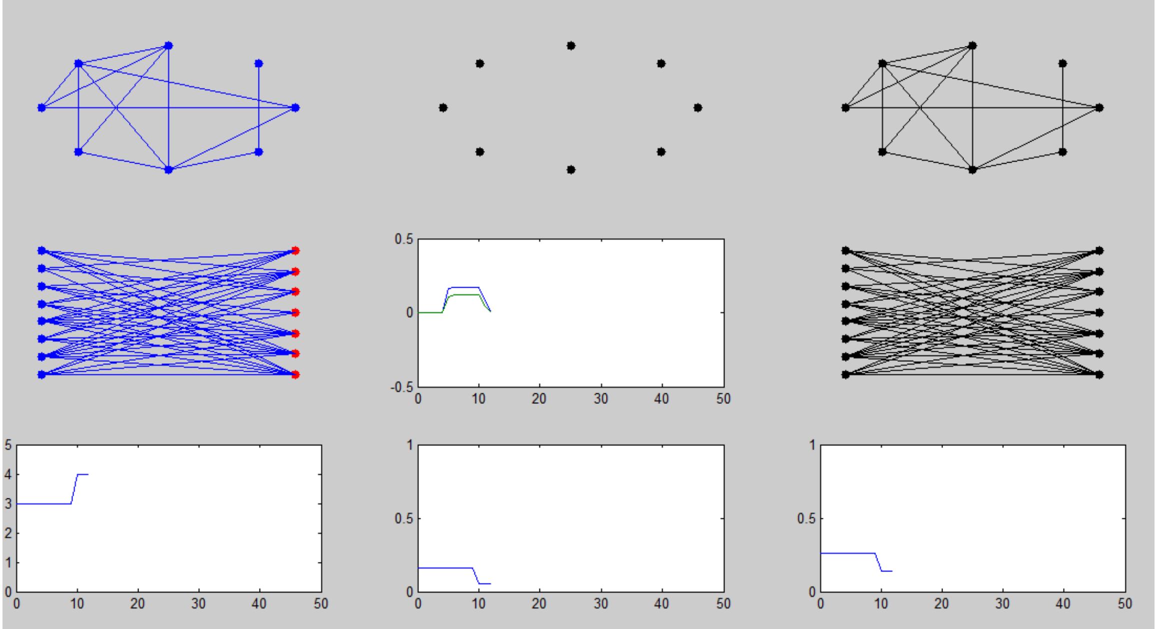


Figure 45 Example Volatile Run Early

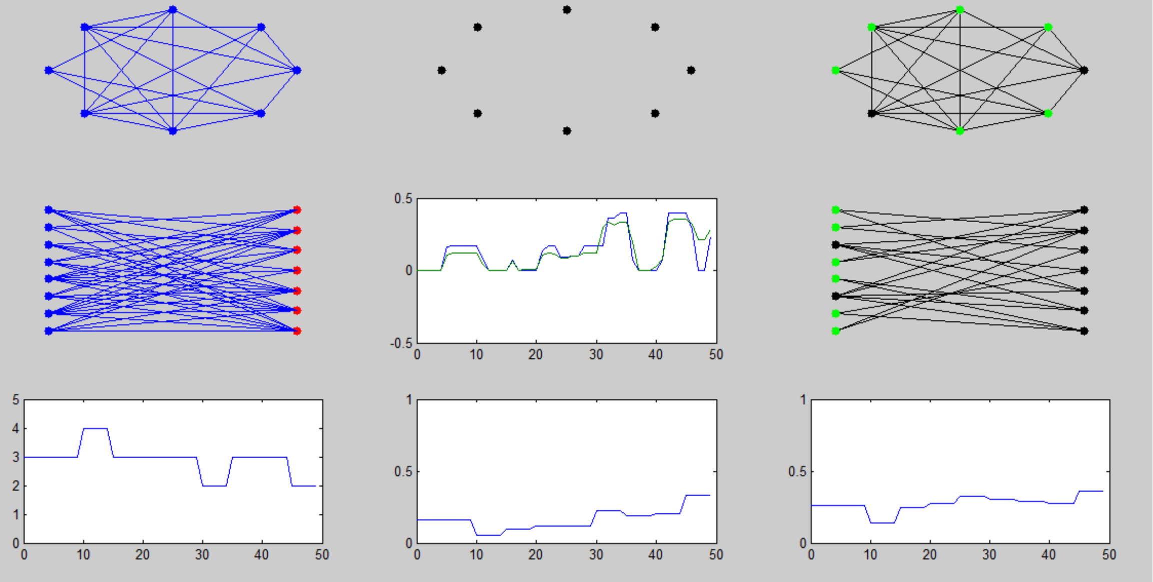


Figure 46 Example Volatile Run Late

The next test run was a test case making the only possible changes being link addition to the system. This was mainly to visualize how this change affected the system and make sure the variables were behaving properly. Such changes should never be de-stabilizing to the system because they only add more information to each agent or make that information more current and up to date. This can be seen in the system performance metrics as well as the network metrics. One interesting item to note is that the Cheeger constant in certain cases can decrease (meaning a move toward greater instability) even when they system should and is only becoming more stable.

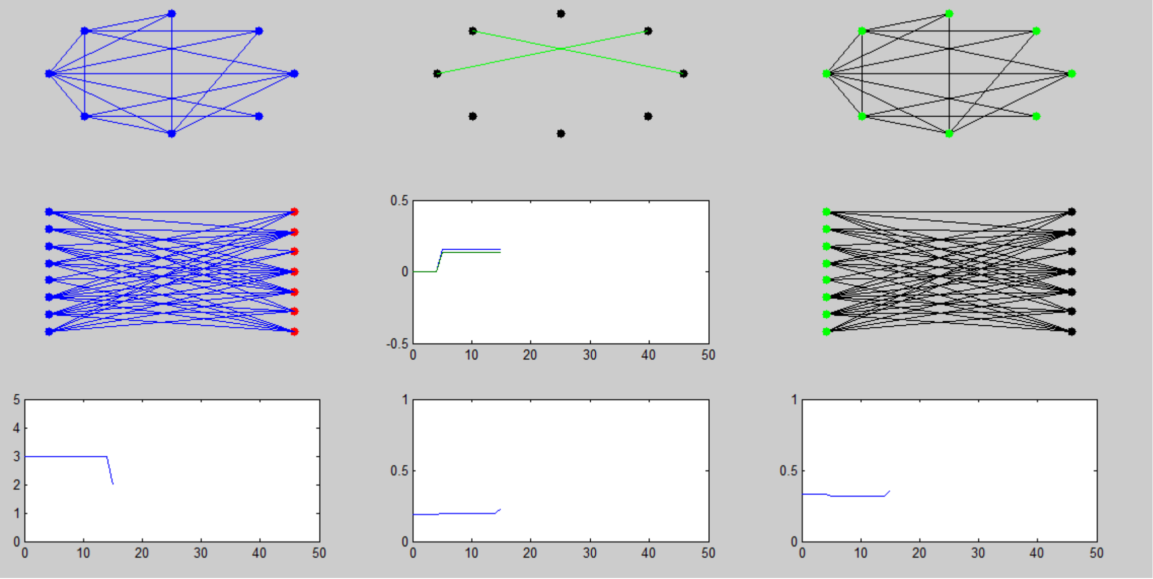


Figure 47 Addition of Links Middle

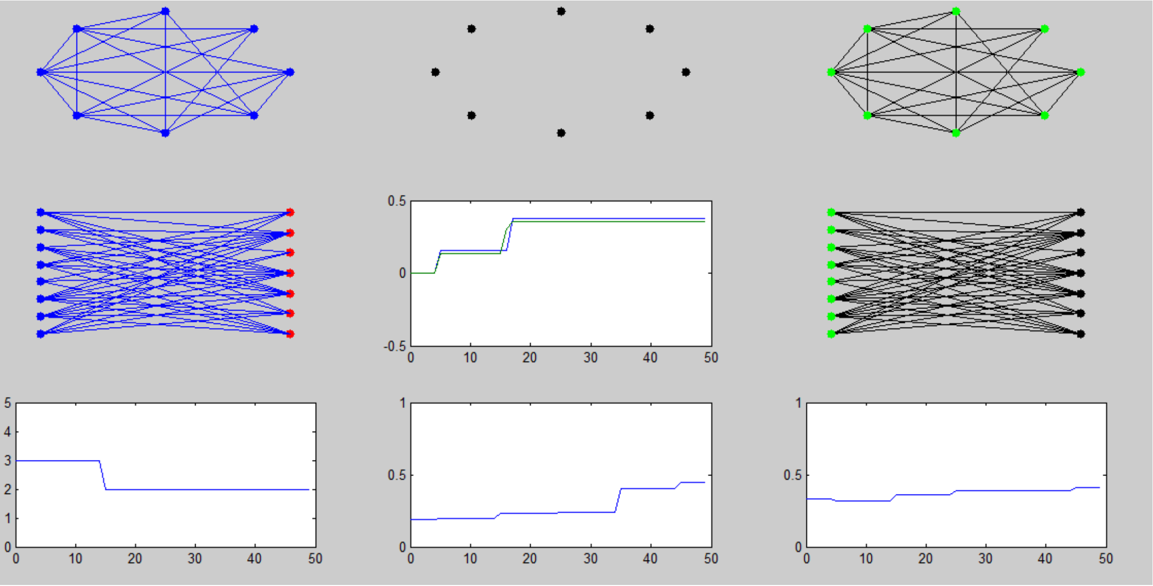


Figure 48 Addition of Links End

CHAPTER 4: CONSENSUS, AGREEMENT ISSUES AND COOPERATIVE CONTROL TECHNIQUE

The purpose of this chapter is to discuss the cooperative control methods developed and implemented in this research. This will focus into two areas, the first being a concern involving information agreement and consensus related issues and the second being the actual control scheme.

4.1 Consensus and Agreement Issues

This piece of the document addresses a critical issue that arose during research and had to be addressed before continuing with the previously discussed network based control methods. This issue is closely related to consensus of the system or the lack thereof. In the original goal of this research, the control scheme would be implemented as given in Figure 49 below. The idea was essentially to base the control scheme on key network parameters and make changes based on that network.

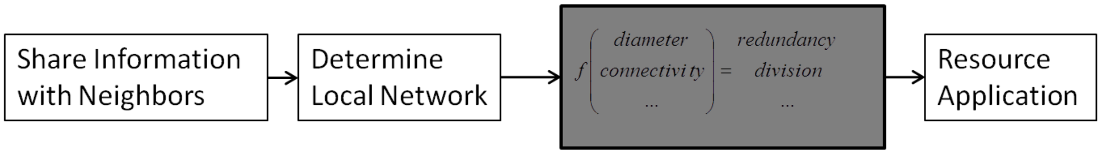

Share Information with Neighbors
Determine Local Network
f (diameter connectivity ...) = redundancy division ...
Resource Application

Figure 49 Original Plan for Flow of Control Scheme

4.1.1 Failures of Information Agreement

To put it simply, the problem with this idea is that it did not work. The system had too many failures to be considered a success, with performance worse than the baseline case. Any changes to the control scheme making it more conservative did alleviate the problem somewhat, but the number of failures was still much larger than desired. This plan had to be modified, and the means in which that modification came was from consensus. Consensus, or the agreement of system information amongst the actors in the system, is needed for good behavior to occur. Without some level of agreement, different actors are trying to complete different goals which may or not contribute to the actual overall goal. This issue has been demonstrated for this specific problem in Chapter 1. The main difficulty of this fact is that a true consensus cannot be reached; the communication lags in the system in addition to the possibility of constant changes automatically create the possibility that certain agents in the system will not have an accurate view of the system. The impossibility of ensuring system agreement is in direct opposition to the desire of some agreement in order to make the control scheme reach its desired goal. The solution to the problem comes by looking at consensus from a different angle; despite the fact that the state of consensus cannot be ensured, is it possible to determine when the system certainly cannot reach consensus? To put it a different way, is it possible to determine when disagreement is more likely or pronounced than other cases? In short the answer is yes; rather than being able to determine whether or not consensus has been reached, it can be determined whether or not a new state has been reached. This document will call this state pre-consensus and what exactly that means will be discussed below.

4.1.2 Pre Consensus

This document will define pre-consensus in the following way: when the system is in a state where agreement on system information is likely, but cannot be ensured due to the fact that all neighbor information will be old. After some time from when a major change occurs, all members of the team should know about that change. The age of this information means that when changes occur, it will take some time for their impact to be known; this fact makes a true consensus impossible. During this state any change can occur and it would take at least one time step for that information to propagate to the rest of the communication network, even in the best case. Though it is not a true consensus, it can still be useful and knowing when pre-consensus is not possible allows for other methods to be implemented. These instances will mark a time when the system must be able to handle a large amount disagreement and possible change. The major source of agreement desired in the system is for the cost matrix. This uniquely defines the assignment which should be made, and as long as there is agreement on it, that assignment will not fail. The other metric of interest is the local network. This is a measure on the system's ability to react to changes, although it is of lesser importance than the cost matrix. If it was possible to ensure that the cost matrix was agreed upon via some mechanism, then a proper assignment could be made regardless of the network structure. In order to understand when pre-consensus is reached or when it is not, the four major changes will be revisited. The way in which each change affects these two variables (and system agreement) will be discussed below.

4.1.3 Major System Changes

- Addition or removal of agents
    - Changes cost matrix (adds a new row)
    - Changes adjacency matrix (adds a new column and row)
    - When changes occur in this way, it will not be seen in time for the next assignment whether it be addition or removal. When agents are added the new agents will not have the information of the other agents, and will require some time to receive this communication (if possible). When agents are removed, possible assigners are removed which would cause system failures. The difficulty arises from the fact that removed agents

are only known to be gone when no communication is received from them. This occurs after an assignment has already been made in their absence.

- Addition or removal of tasks
    - Changes cost matrix (adds a new column, or removes an existing one)
        - By definition a proper system requires each task to be assigned by at least one task.
    - No changes in adjacency matrix
    - When changes occur in this way they are noticed in time for assignment by the specific agent, but not able to be communicated to neighbors before that assignment. When a new task is detected, the detecting agent(s) can respond and assign that task before any failures occur. The idea is the same for newly created compatibilities with existing tasks. When a new compatibility arises; it is treated the same as a new task being created in that the corresponding agent can respond immediately before the next assignment. When tasks are removed, other assignments become available, but this also causes a difference in information such as cost and that information must be transferred to the rest of the network before others know of this change. In summary, changes of this type can be dealt with by agents affected by them before the next assignment takes place.
- Network modification via link changes

  - Does not change cost matrix
  - Changes adjacency matrix (via entries changing between zero and one, does not change size)
  - This change alone cannot cause any direct problems in the assignment as the cost matrix itself remains unchanged. However it can easily make the system more or less stable and change its ability to react to future changes of other types. This change will not be known before the next assignment takes place, and it must also be communicated to be known by the rest of the network. The biggest problem with this change is that it initially appears like an agent was removed when a link is removed. An issue of the system is fact that the removal of an agent is only known by the lack of communication from that agent. Therefore an initial lack of communication will carry no information as to whether the communication failed or that agent is no longer part of the system. This fact confounds a relatively minor change with perhaps the most impactful one.
- Cost changes
  - Changes cost matrix
  - Does not change adjacency matrix
  - This change directly causes disagreement of information, and if that change is dramatic enough will cause problems in the assignment process. Changes in the cost matrix are continuous, unlike the previously discussed

system changes which are discrete. This change will be known before a change occurs for the agent(s) involved with the change, but the rest of the agents in the team will not until it is communicated.

It is clear from the numerous failures without considering pre-consensus that the majority of those failures are happening with the system is not in pre-consensus. It became clear that the amount of redundancy during this phase can help prevent failures but cannot ensure that failures will not occur. The only way to ensure that the system will not fail when not in pre-consensus is for the affected agents to transition to maximum redundancy. This state of maximum redundancy is equivalent to the baseline case, and as stated in previous chapters, it is impossible for the baseline case to fail as long as the problem remains well posed.

The next problem to tackle is when the system returns to pre-consensus after changes occur. In all likelihood this will have something to do with the diameter or radius of the network as those variables directly relate to the amount of time it would take information to propagate through the system. Knowledge of the change should be allowed to propagate throughout the system with certainty before entering pre-consensus. In order for this happen, agents must wait a number of times steps greater than the diameter of the system while in this conservative state before pre-consensus is reached and other methods implemented. While it may be possible to enter pre-consensus in less than that amount of time, it is desired that failures be eliminated with certainty even with the performance reduction associated with staying in the baseline case longer than needed. A lower performance case which minimizes uncovered tasks is more desirable than a higher performance case which does not. When a change occurs it may be possible that changes are also occurring in other parts of the network, and allowing those to propagate throughout at the worst case would be a time equal to the diameter of the network plus one time step. To demonstrate information propagating through a communication network, a basic sample structure is given below in Figure 50.

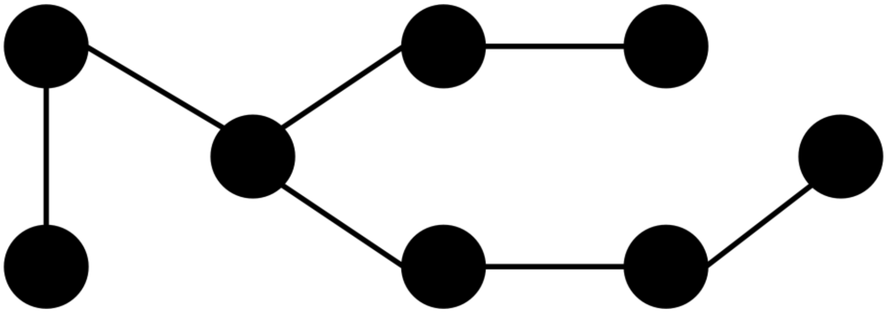


Figure 50 Notional Network

In the first case, a single change will occur at the central node of the system as shown in Figure 51 below. This change will propagate throughout the rest of the network as seen in Figure 52, Figure 53 and Figure 54.

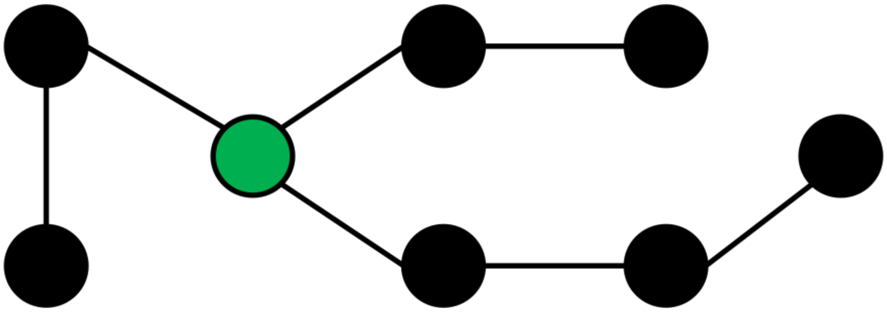


Figure 51 Notional Propagation Step 1

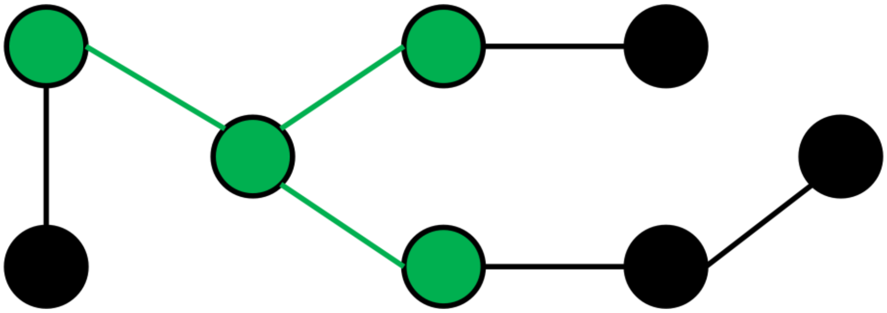


Figure 52 Notional Propagation Step 2

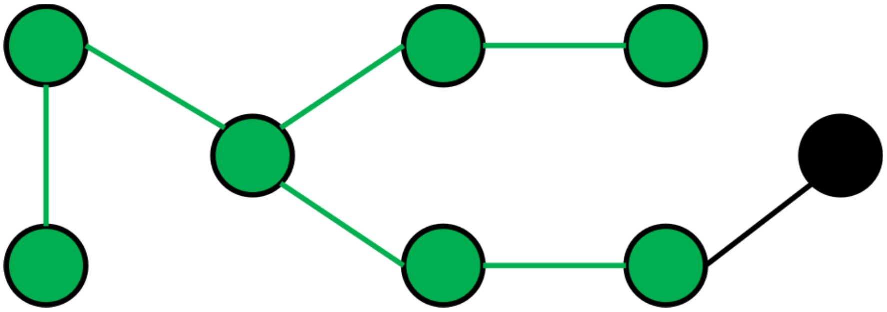


Figure 53 Notional Propagation Step 3

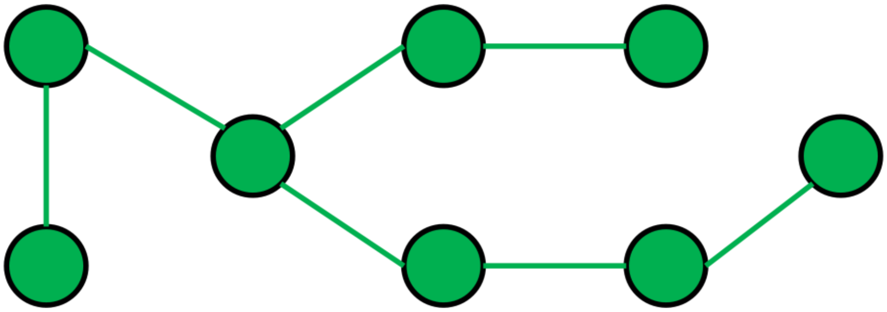


Figure 54 Notional Propagation Step 4

Now envision another change occurring to Figure 50, but in this case it occurs at the end of the network rather than its center. All but the final communications are given below in Figure 55, Figure 56, Figure 57, Figure 58, Figure 59 and finally the system will end in Figure 54.

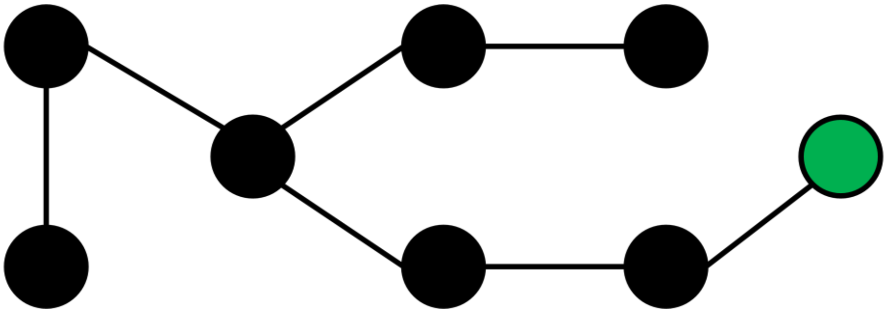


Figure 55 Notional Propagation 2 Step 1

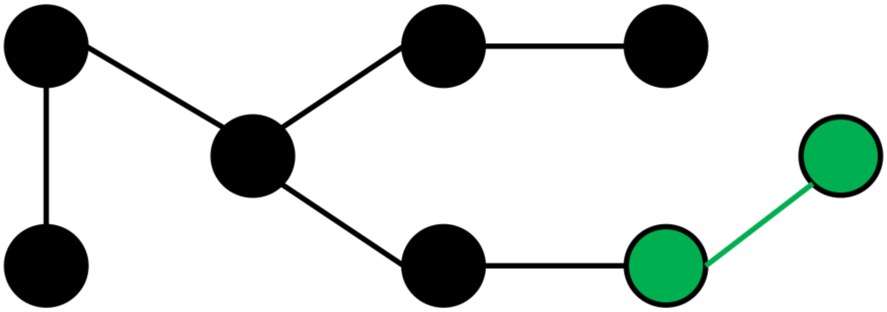


Figure 56 Notional Propagation 2 Step 2

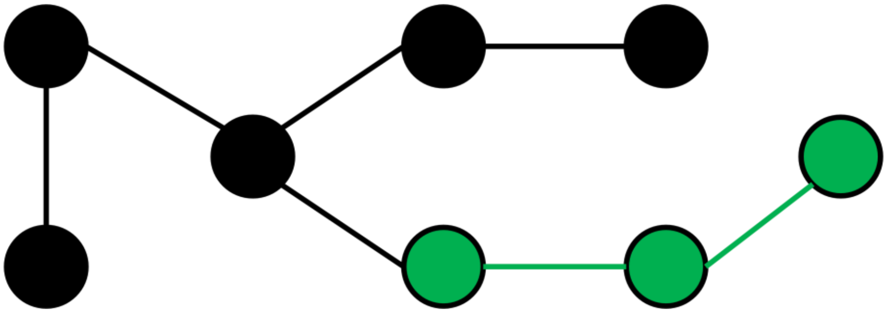


Figure 57 Notional Propagation 2 Step 3

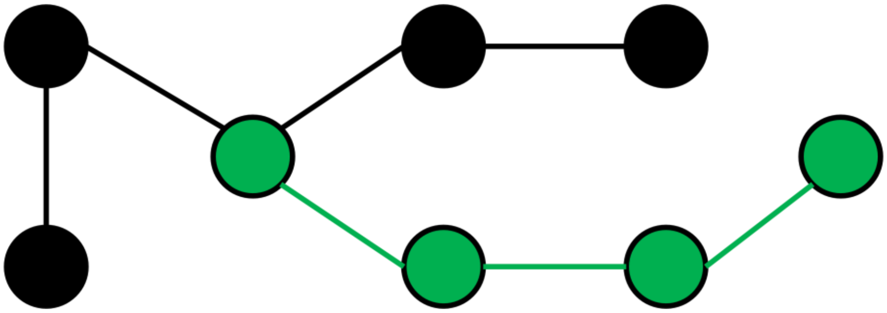


Figure 58 Notional Propagation 2 Step 4

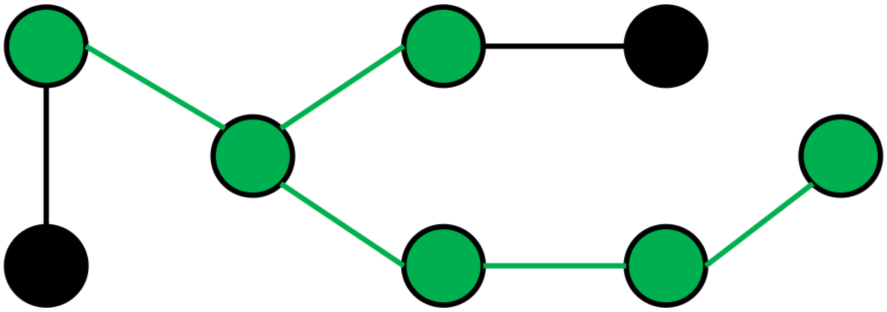


Figure 59 Notional Propagation 2 Step 5

If any change happens independently than it is known exactly how much time it will take for all other agents to know of the change. This is based on network structure. It can be agreed upon in the meantime that the entire system should enter pre-consensus together at this time. The problem is that changes do not occur independently, and entering pre-consensus too early can cause failures. Rather than set an agreed upon time in which pre-consensus takes place, each agent should wait for an amount of time equal to the diameter to enter pre-consensus. This is a more conservative strategy, but will help ensure that multiple system changes which occur while waiting for pre-consensus do not increase the chance of uncovered tasks.

The reason agents enter an assignment scheme equal to the baseline case when not in pre-consensus is because such a state will prevent failures regardless of the system information and its changes. A scheme to keep the system in the most conservative state when not in pre-consensus is given below in Figure 60. This will help ensure that when major changes occur or the system is rapidly changing, an aggressive control scheme will not be used. This is needed in order to prevent such aggression from causing failures when there is not an acceptable level of information agreement.

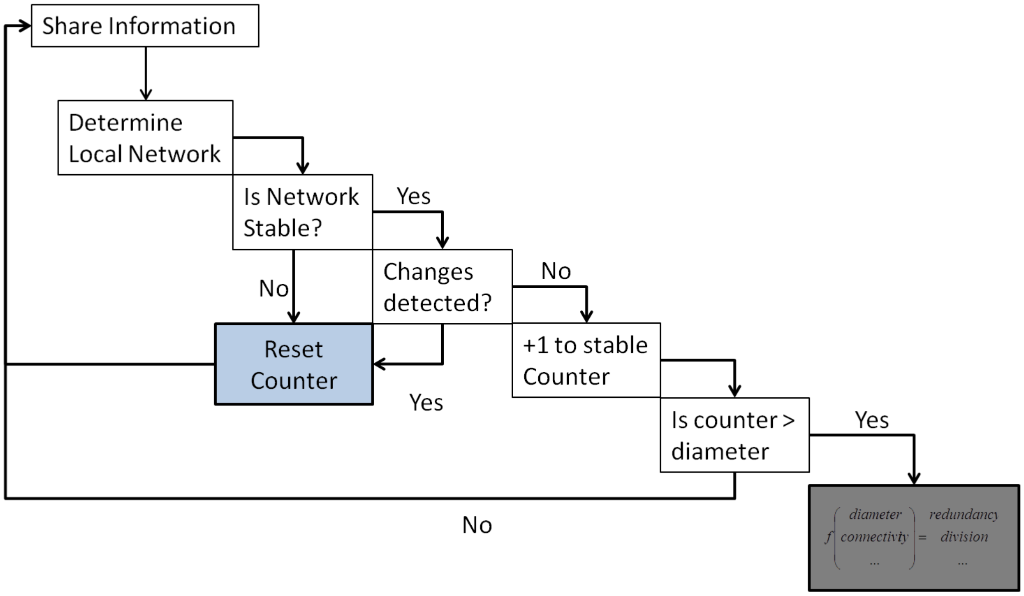


Figure 60 Flow of Pre-Consensus Determination

With this new implementation, the control scheme shown in Figure 49 is modified to that shown below in Figure 61. This modification comes with some benefits in terms of neutralizing certain changes, but also brings some other behavioral changes to the final control.

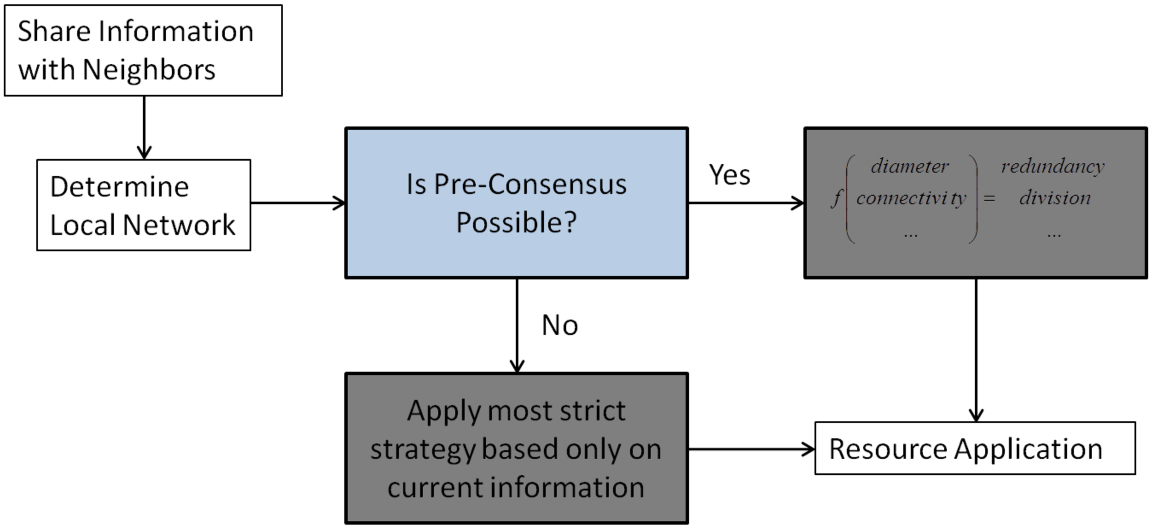

Share Information with Neighbors
Determine Local Network
Is Pre-Consensus Possible?
Yes
$f\begin{pmatrix} diameter \\ connectivity \\ \ldots \end{pmatrix} = \begin{matrix} redundancy \\ division \\ \ldots \end{matrix}$
No
Apply most strict strategy based only on current information
Resource Application

Figure 61 Modified System Control

4.1.4 Implications of Pre-Consensus in Control

The effects of this change are twofold, with the first set being positive and the next being neutral to negative. The greatest benefit of this addition is that it can neutralize the following system changes and ensure that failures will not occur as a result of them:

- Addition or removal of tasks
    - When new tasks are added, those that can be assigned to it immediately shift into the conservative state, when tasks are removed that changes the shared information enough that those that notice the change will also shift into the most conservative state. In this way at least one agent who can be assigned to these tasks will be.

- Modification of links
    - When new links are added to the system this is in fact a good thing as it does not reduce the stability of the network. When links are removed it is treated the same as removal of a neighboring agent because it is impossible to know that did not happen until later time steps. In addition, the change in network metrics caused by this may be enough to change the control scheme, which could also benefit from a shift toward conservation before implementation.
- Change in cost*
    - If any change in cost is considered a major change than failures from it can also be eliminated. This is for the same reasons as the addition or removal of tasks; the changes in costs affect agents who are able to respond to those changes and cover possible effected tasks until the change is able to propagate and be known to the entire system.
    - The reason this change is marked while the others aren't is due to its special nature. The other two changes previously mentioned are discrete changes in the system, and even the smallest changes can be enough to cause failures if not handled properly. The changes in the cost matrix are continuous changes of real variables. While even the smallest change to this matrix can cause a failure, it is likely that only major impacts will cause failures. It may be possible that some changes which are small or

gradual can be handled via redundancy rather than requiring a complete reset. By causing a system reset at all times, the control behavior will suffer for reasons given below. Other means to handle the cost matrix will be discussed in later sections. However even if those techniques do not prove successful is possible to marginalize all cost changes with a hard reset for any cost change.

Of the four major changes which can occur two can be handled without resulting in system failures. The third can be handled, but it may be more attractive to tackle that problem in a different way. Only the removal of agents cannot be handled, and that problem must be marginalized through the control scheme via redundancy. However any change of this type while the system is already in its conservative baseline state will not cause failures, so that is also an attractive trait.

All of those benefits do not come for free however, and the system can no longer be promised to be strictly greater in performance to the baseline case. The reason for this is that entering pre-consensus is beyond the control of the system; entering pre-consensus is a reaction to system changes based on the network structure. If changes occur at a more rapid pace than the system can propagate information, pre-consensus can never be reached. This idea makes sense; if the information acted on by a teamed system can never be agreed upon than trying to do anything more than the most basic scheme could very well cause system failures. Knowing that the behavior of agents when not in pre-consensus is beyond the action of the control, the goal now is to devise a method to neutralize the last two system changes as they occur during pre-consensus. This is the only time in the current scheme that the system can fail.

Another interesting note via the implementation of pre-consensus is the meaning of the normalized metrics. In both normalization schemes, the metrics have a clear meaning to indicate when the system is in the baseline case (a value of zero for total normalization, a value of one for baseline normalization). When the system cannot enter pre-consensus, the corresponding values of the metrics of interest will reflect this fact. All values will be zero or one (total or baseline) for each time step of the simulation.

4.2 Cooperative Control Technique

The purpose of this section is to describe the implementation of the cooperative control technique. The basic idea of the control is to convert the key system metrics (which will be discussed and determined) into the main control parameters: desired redundancy and resource distribution variable. This involves determining what key network parameters best represent the system, and then finding a scheme to convert those system metrics into the corresponding control values. Those two components will make up the remainder of this chapter, with some special considerations and issues discussed afterwards.

4.2.1 Determination of Key Network Parameters

The first step to designing the control is to decide which networks metrics are chosen. The desired metrics should be able to capture in some way the stability of the communication network. There is no single network metric which can fully capture this idea out of those studied previously, but the closest is the diameter. Essentially the collection of metrics should give some understanding to how compact the network is, and how likely it is for that to change. These metrics should have clear differences in value between networks of different stability, and have normal well behaved characteristics such as being monotonic for networks going solely from unstable to more stable or vice versa. Three tests were conducted in order to demonstrate this; the first is demonstrated progressively in Figure 62 below.

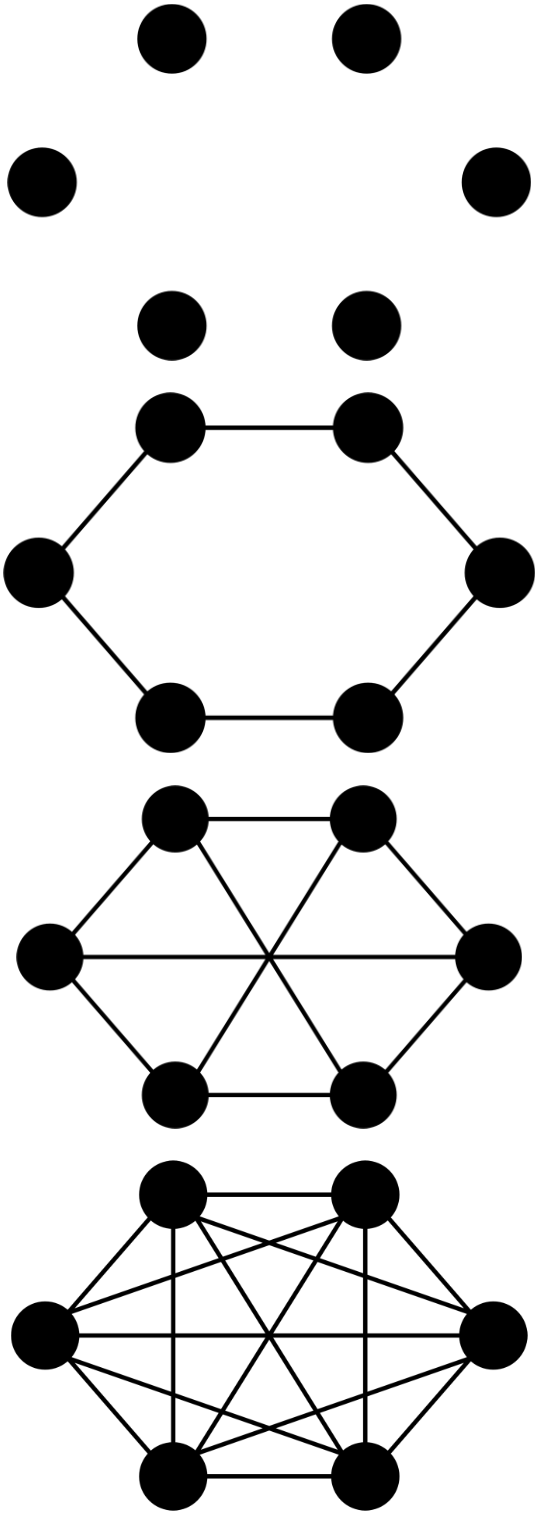


Figure 62 Network Build Up via Link Addition

This test gives an example of a network becoming more stable while holding at a constant number of agents/nodes. Each network increases in compactness as compared to the previous one. The second test is one of decreasing stability while increasing the number of agents/nodes in the network. This is progressively given below in Figure 63. The final test will be discussed after as it gives some idea of the dynamic nature of these variables. It is possible for a metric to pass the first two tests but not the final test and it still be useful in the control. The dynamic test of the metrics is used when determining whether the system is in pre-consensus or not and how the system progresses in that way, while the static cases will be used to quantify what control variables (redundancy, distribution) should be applied to the system.

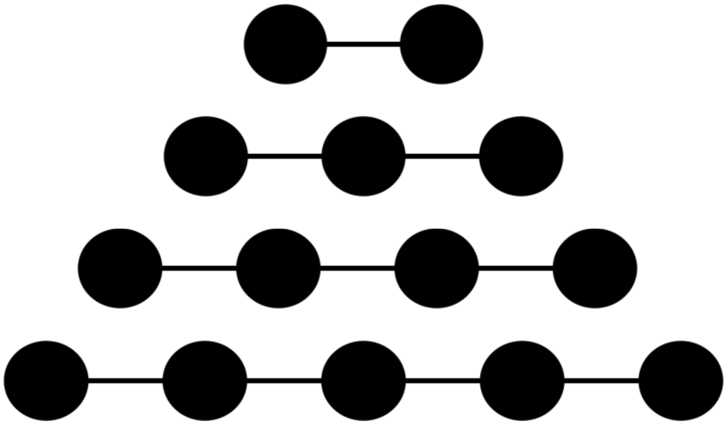


Figure 63 Build Up of Line Network via Addition of Nodes

Each of the metrics previously discussed is presented for both of these tests and will be discussed. The desired result is to have metrics which show movement in different directions for the two cases, as these two cases show improvement and reduction in stability. For example, if a metric increases for the first test, it should decrease for the second. If it follows the same behavior for both, than that metric does not capture the stability of the network effectively. Next, the variables should be monotonic as the improvement or reduction is similarly monotonic. Finally, even if it is small, there should be some difference in value between the different cases; a constant value will not give a good demonstration of the differences between the cases. While it is not a requirement, metrics which have clearly defined bounds will be of greater value than those that do not. The first case is shown below in Figure 64. Once again, the three criteria are presented below:

1. Show different behavior for the increase in compactness (Test 1) vs the decrease in compactness (Test 2). IE one should increase and the other should decrease.
2. Have monotonic behavior. Cases represent increasing or decreasing compactness, metrics should also represent this.
3. Values should not be constant over entire range of test. Cases were designed to spread over a wide range of networks, a constant value gives no meaning to the differences betweent these networks.

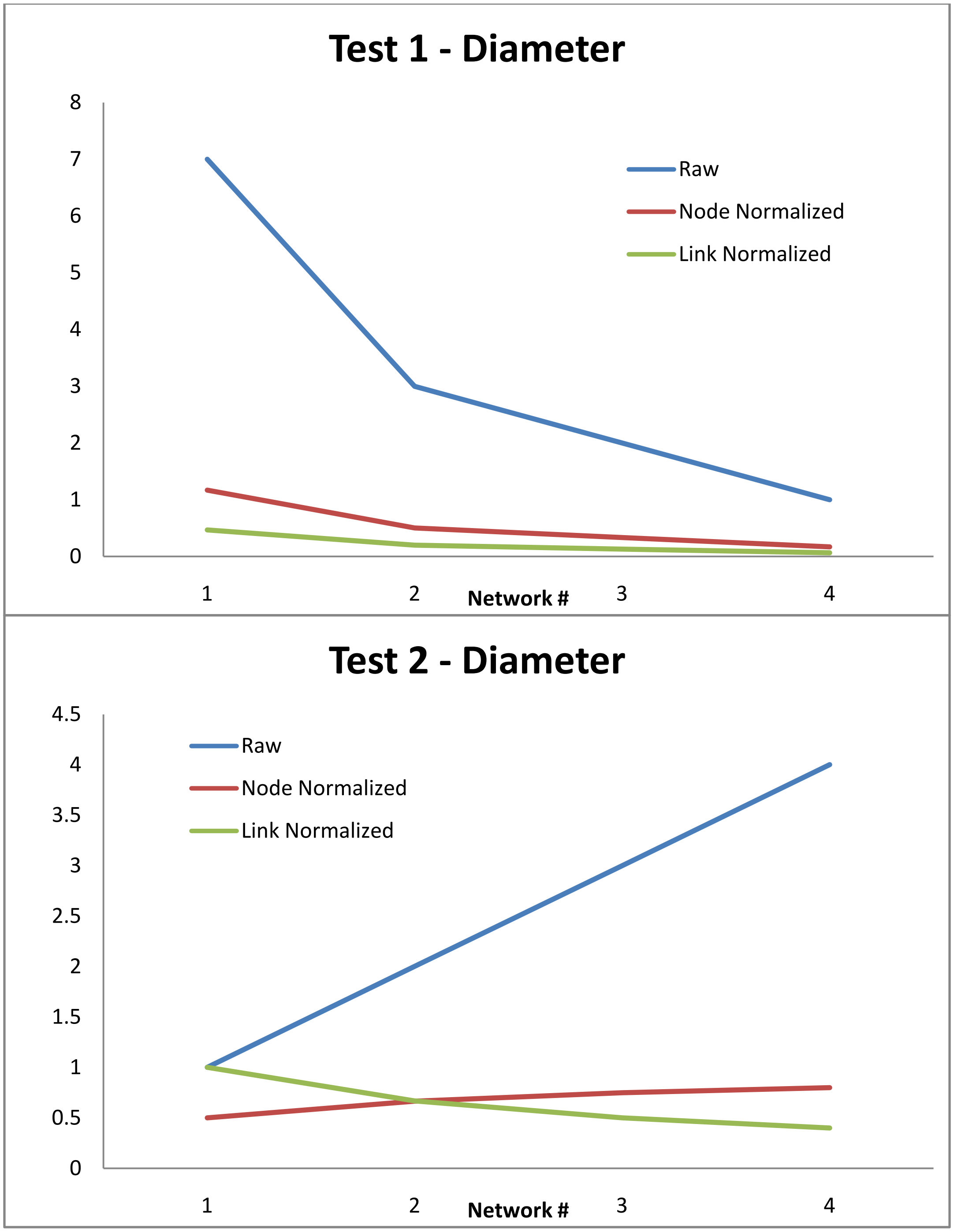
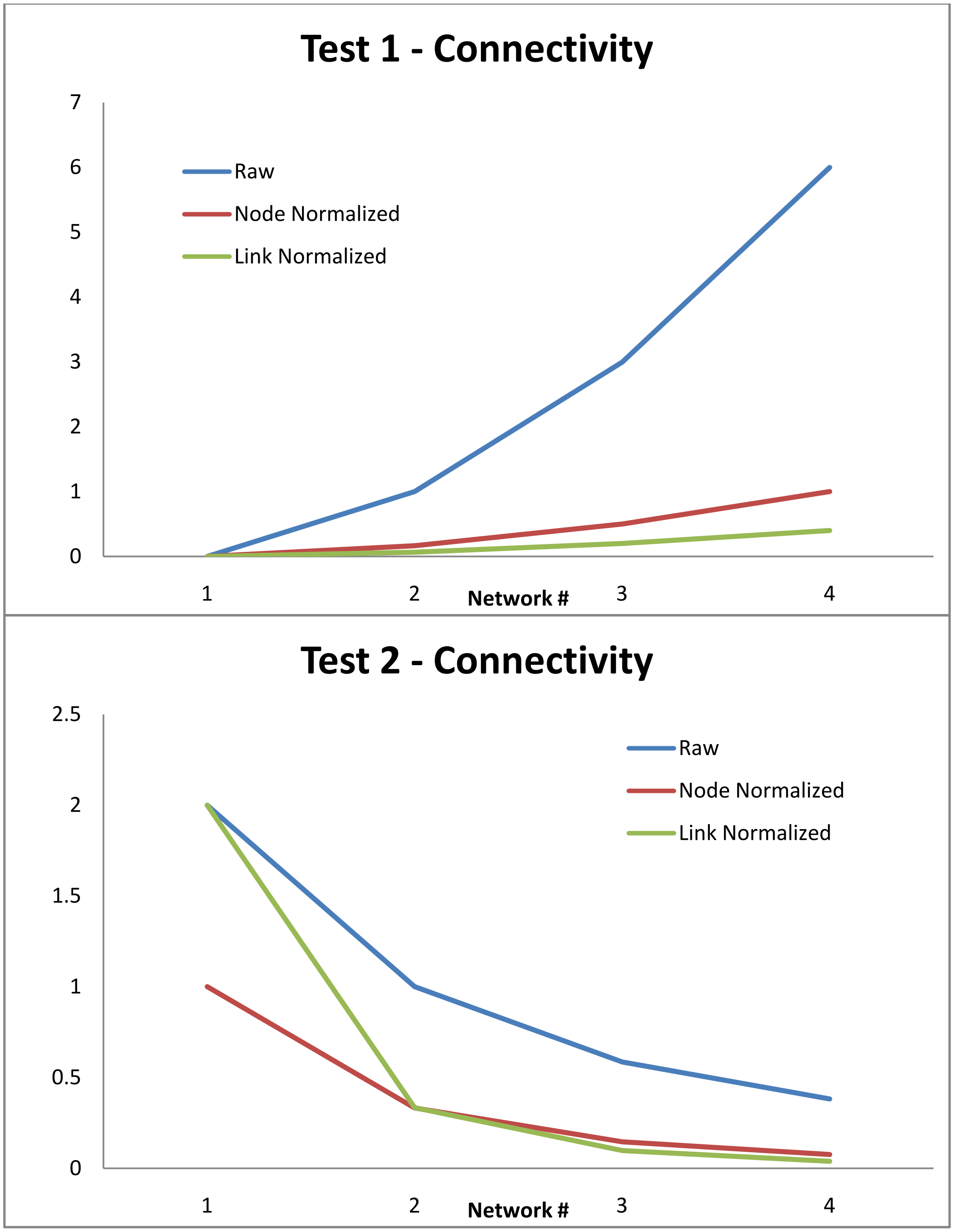


Figure 64 Diameter Comparison of Test Cases 1 and 2

The diameter meets all three criteria discussed above; however normalization via the total number of links fails the first criteria. The raw and node normalized cases show different facets of the problem; the raw case giving the exact value needed to determine the time it would take for communication to occur, and the normalized value gives an idea to the diameter as compared to the size. Any value over one in the normalized case is a disconnected network, while a value less than half gives a relatively stable network despite its size. The next metric discussed is the connectivity and is given below in Figure 65.



Test 1 - Connectivity

Test 2 - Connectivity

Figure 65 Connectivity Comparison of Test Cases 1 and 2

The connectivity shows good performance for all three criteria. The only difference in these cases is based on the normalization. For this issue, the node normalized case is the best and will be brought forward. The raw case shows good performance but has a slightly different meaning than what is desired. The connectivity ranges from zero in an unconnected graph to the number of nodes in a fully connected graph. This is not attractive because comparison between graphs of different size would become an issue. Normalization via the number of nodes solves this problem, and in such a case the most attractive value of the connectivity would be one. The next metric(s) discussed are the two approximations of the Cheeger Constant, and the performance vs. the two test cases is given below in Figure 66 and Figure 67.

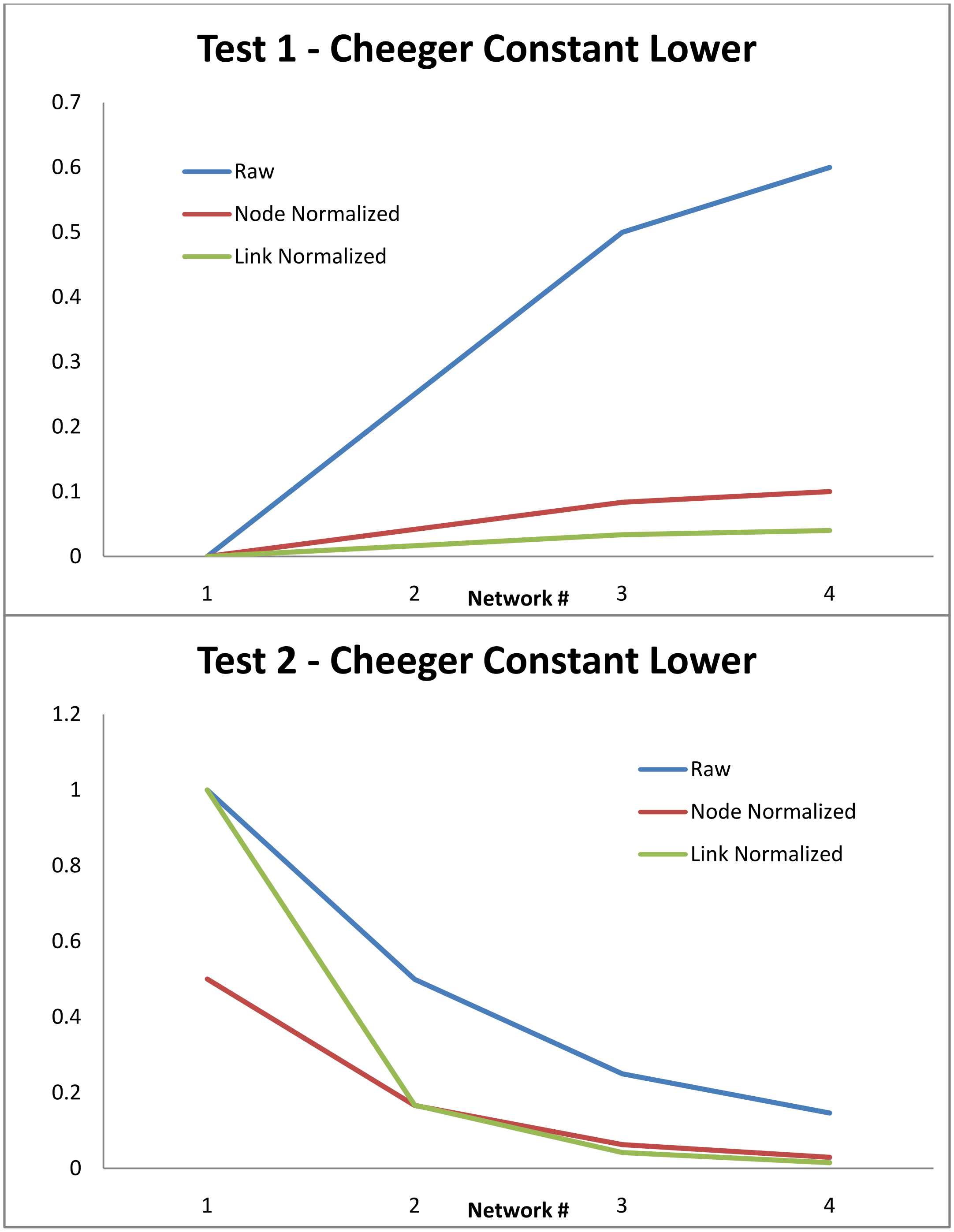


Figure 66 Cheeger Constant (Lower Bound) Comparison of Test Cases 1 and 2

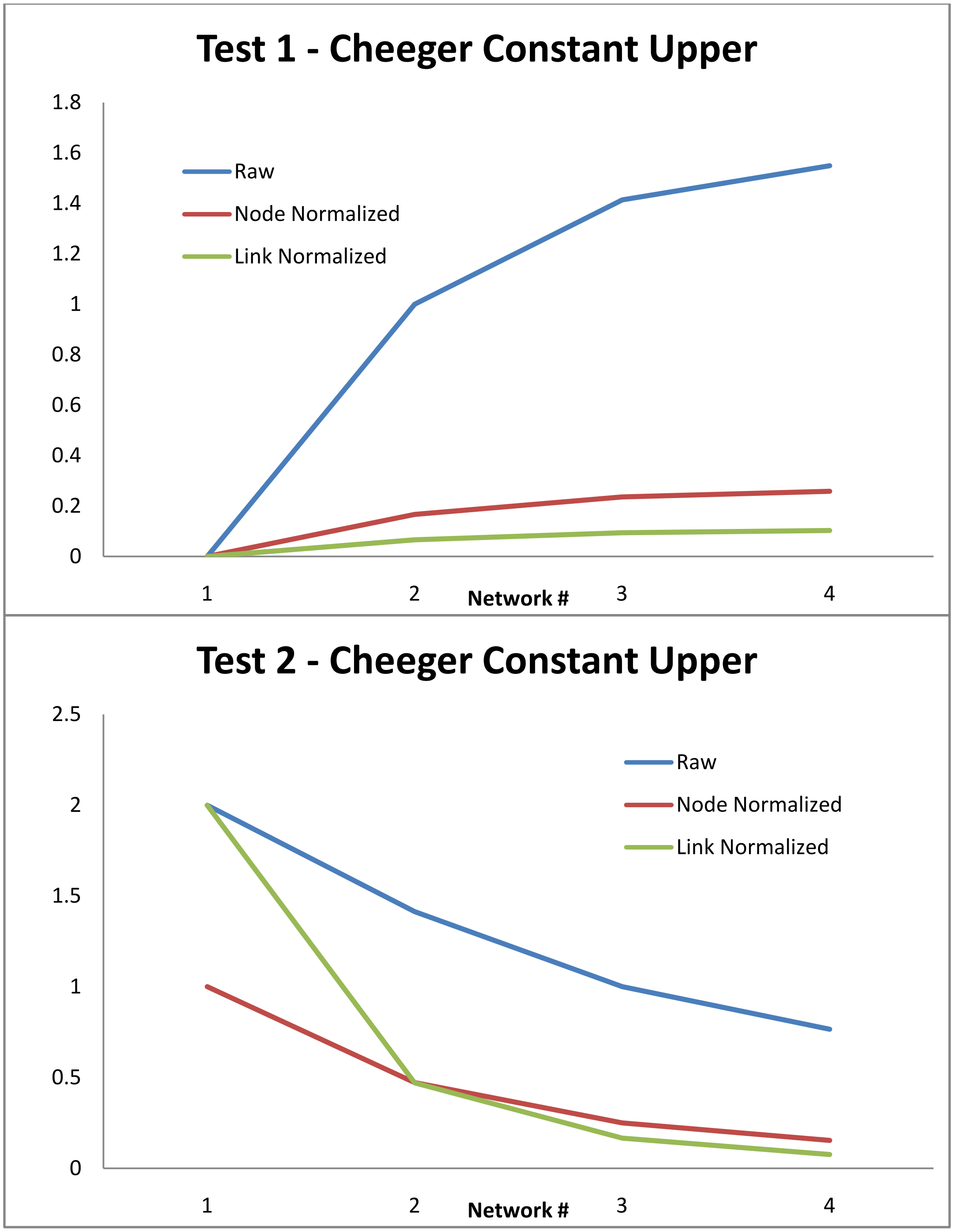


Figure 67 Cheeger Constant (Upper Bound) Comparison of Test Cases 1 and 2

Both variables meet the desired criteria. The normalization method is not as clear in this variable, as neither bound has a simple maximum based on network parameters. The link normalization gives some interesting performance in both cases, but it seems somewhat large in smaller networks. It is difficult to say which is best based on these tests, therefore the raw value will be used. This variable is not as strong as the others previously discussed, but does give some idea that the system is sparse when the value is small. The raw case gives good differences between the cases which is attractive as well. Finally, the overall performance of each approximation seems about the same vs. the lower and upper bound. As such the lower bound is chosen as neither one seems to have any advantages or disadvantages over the other. The final metric, the number of links in the system, is given below in Figure 68.

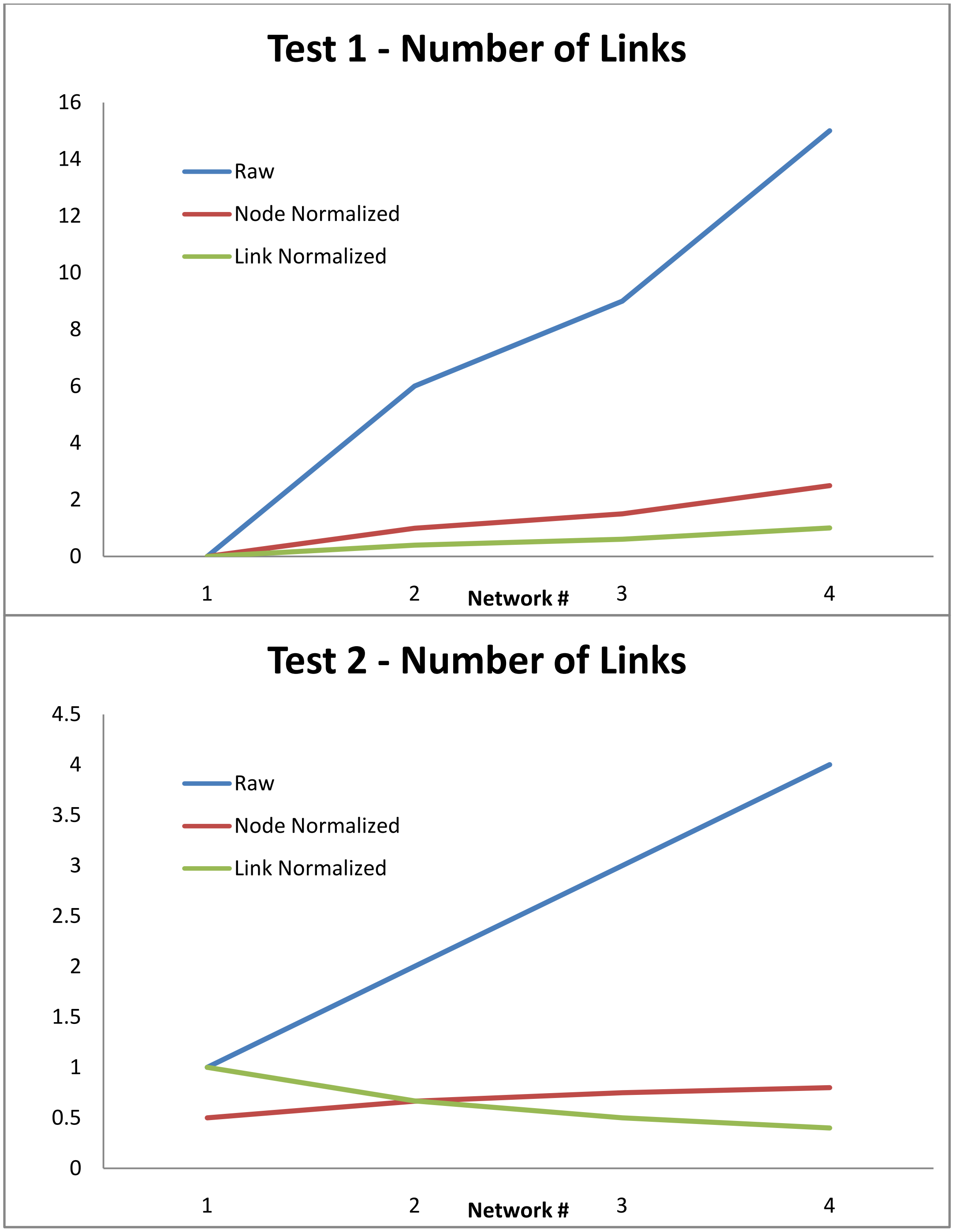


Figure 68 Number of Links Comparison of Test Cases 1 and 2

This metric does not satisfy the first condition given above. For the raw and node normalized cases, the values increase for both cases even though that should not be the case for a metric to represent the system. The link normalization passes this test but has another flaw; essentially it is a more crude representation of what is already given by the node normalized connectivity. The link normalization gives a simple ratio of the number of links to the total number of links while the node normalized connectivity considers grouping of nodes which are linked and will be lower or equal to the link ratio. This makes the connectivity a better measure of the network and a more conservative metric which will be useful in the control.

This concludes the metric analysis for the static metrics of the system. The results are briefly summarized below for each metric:

- Diameter – The best of all metrics. It may be possible to use this metric alone, however it is also the most granular of all metrics as it is a discrete value rather than continuous.
- Connectivity – Gives some idea of the number of links in the system as well as which links exist, giving more importance to links to lesser connected areas. Unlike the diameter this variable is less discrete and can take a larger range of values. In addition this value is bounded and each bound gives clear physical meaning.
- Cheeger Constant – Another "sub-metric" when compared to the diameter. Gives an idea of stability analogous to the connectivity in that the more connected the

network the higher the value but its meaning is a little different. Lower values imply that an otherwise stable network is susceptible to large impacts with key losses in nodes or links. Unlike the connectivity, this value is not bounded from above and does not imply as clear a physical meaning.

- Number of links – May only serve as a surrogate value of the connectivity. Does not have good properties for the other criteria.

This concludes the static metric selection for the control scheme. Next the dynamic case will be discussed.

4.2.2 Dynamic Metric Consideration

This test involves taking a network in which one node is connected to the rest, and each other node has no connections other than this one. From this point each node is sequentially connected to the rest in order until the network is complete. This is shown below in Figure 69, Figure 70, Figure 71, Figure 72, Figure 73, Figure 74, Figure 75, Figure 76, Figure 77, Figure 78 and Figure 79. This is to demonstrate some interesting behavior in some of the metrics and to determine their use for measuring the immediate impact of changes on the network and whether those changes move the network in a more or less stable direction. This is used in the pre-consensus analysis as a quick way to measure changes to the system. As the control is conservative, this gives an easy way to switch to the more conservative control when needed.

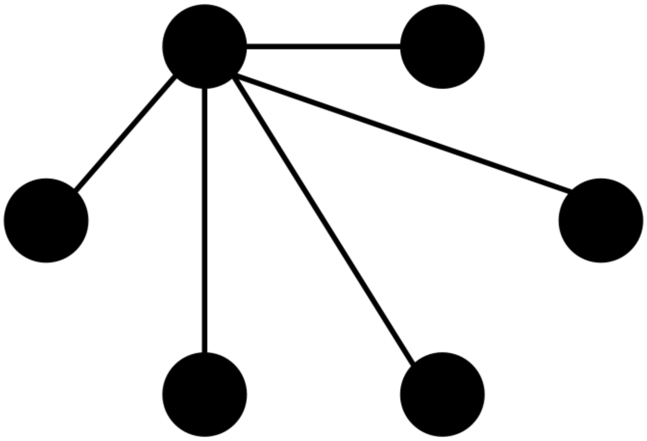


Figure 69 Network Build Up Phase 1

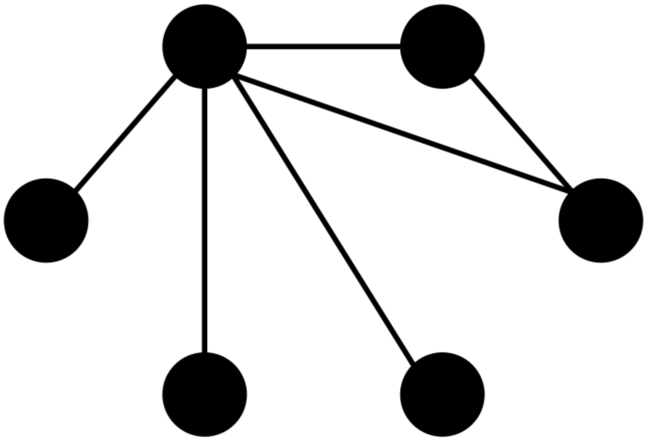


Figure 70 Network Build Up Phase 2

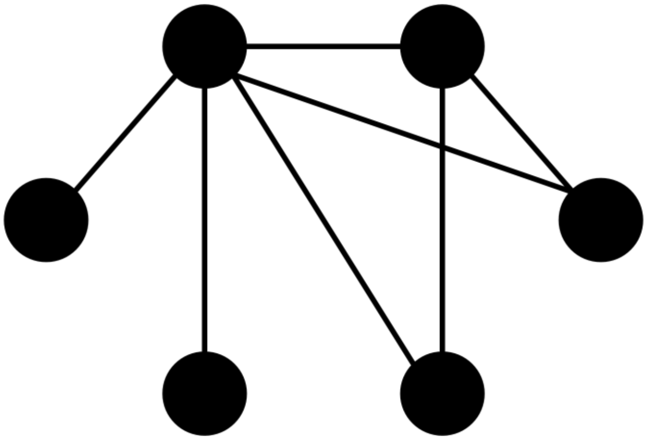


Figure 71 Network Build Up Phase 3

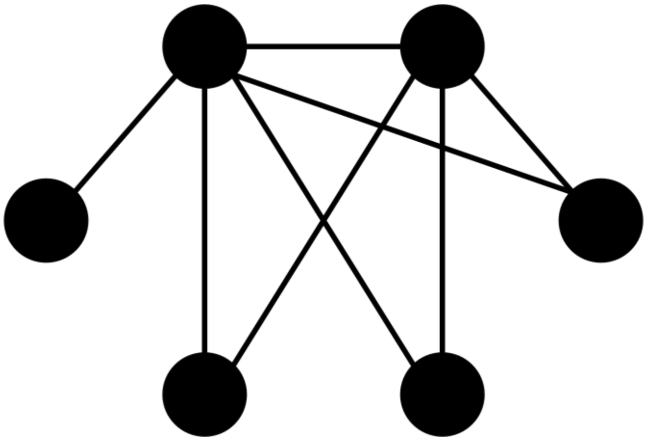


Figure 72 Network Build Up Phase 4

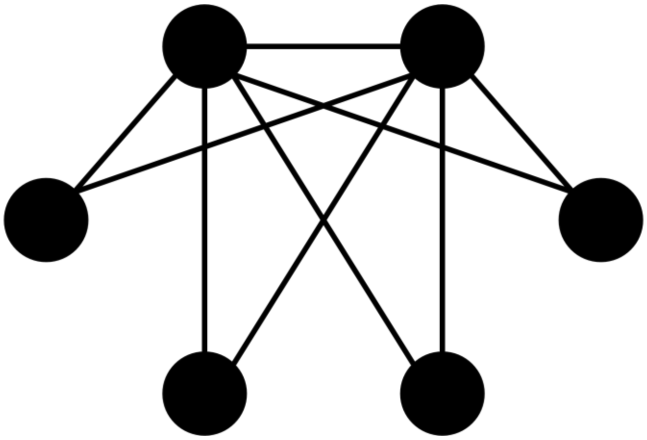


Figure 73 Network Build Up Phase 5

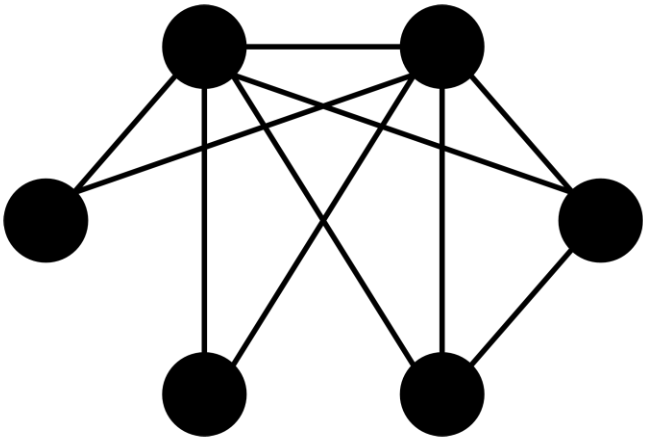


Figure 74 Network Build Up Phase 6

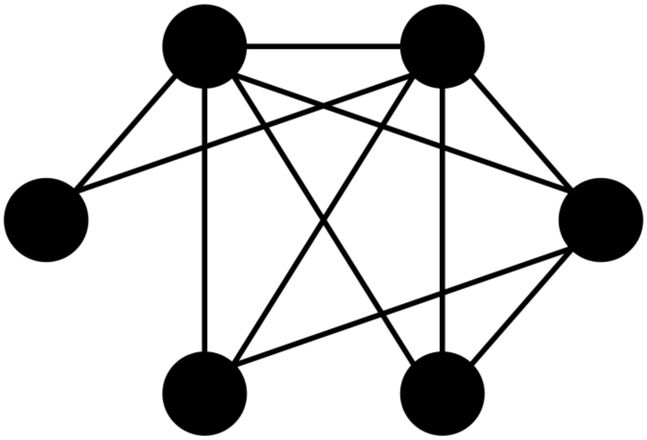


Figure 75 Network Build Up Phase 7

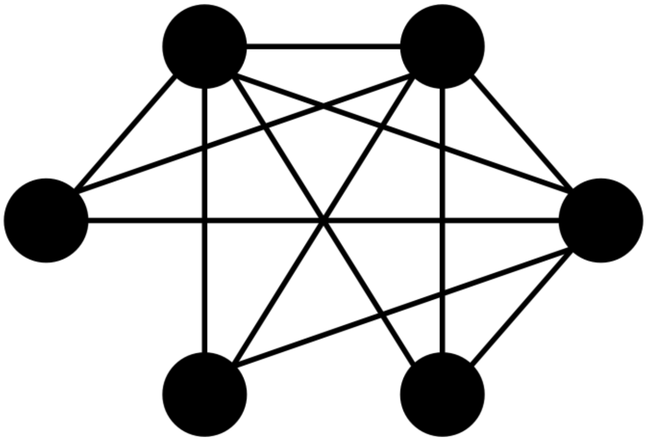


Figure 76 Network Build Up Phase 8

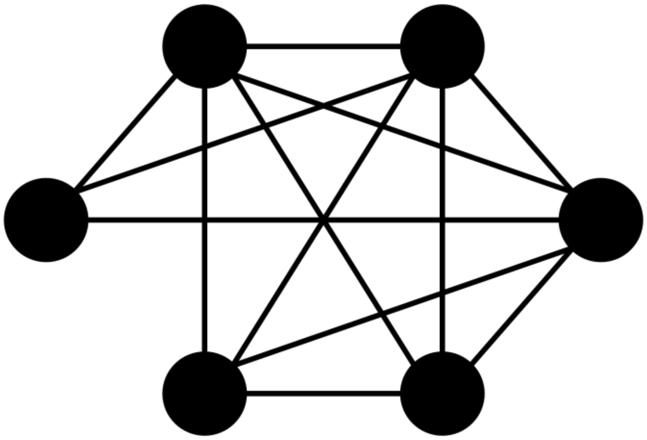


Figure 77 Network Build Up Phase 9

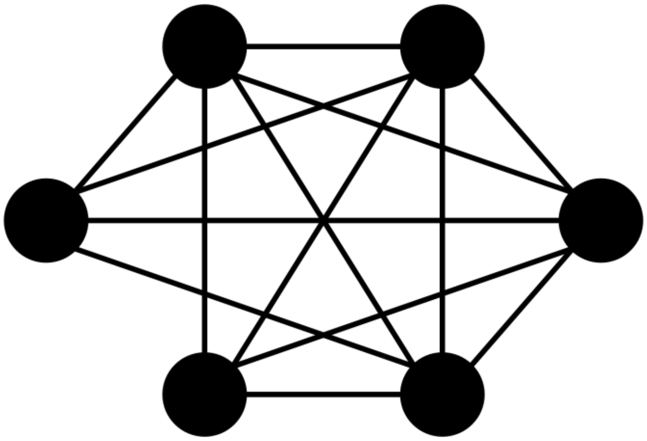


Figure 78 Network Build Up Phase 10

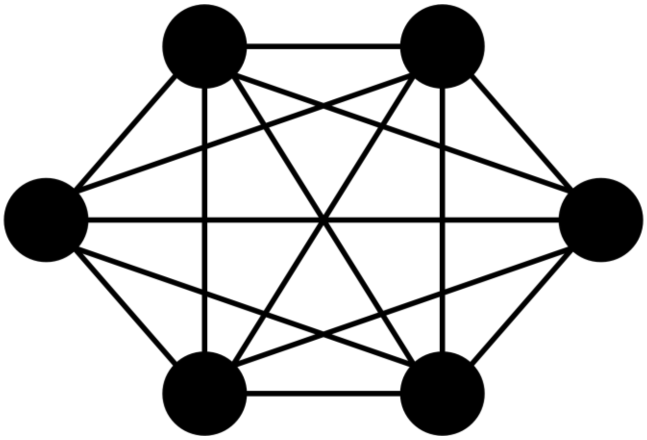


Figure 79 Network Build Up Phase 11

For this test case each of the down selected metrics from before are analyzed. The first of those metrics is the diameter and is given below in Figure 80. For each normalization scheme the behavior meets the standards from before, IE that of monotonic behavior for strictly destabilizing or stabilizing behavior. Another thing to note from this graphic is the limitation of the diameter. Despite great changes to the above network the value of the diameter does not change until the end. This is not always the case, but it is certainly something which needs to be considered.

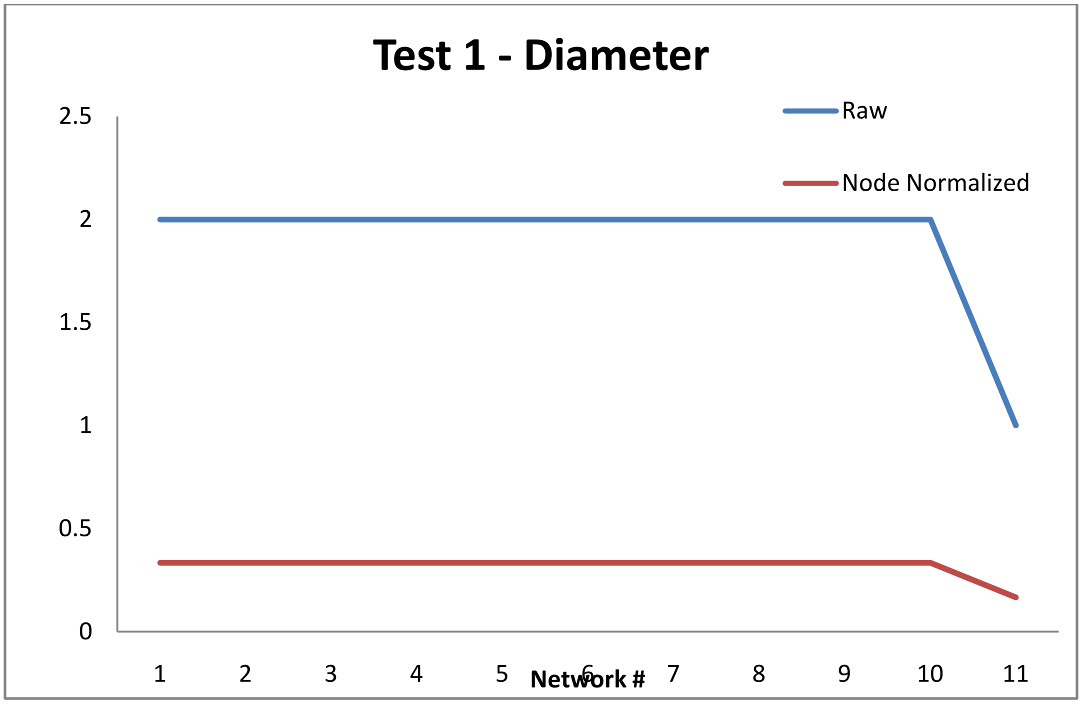


Figure 80 Test 3 Results: Diameter

The next metric analyzed is the normalized connectivity, which is given below in Figure 81 below. As with the diameter, the behavior of the normalized connectivity meets the desired standards or monotonic behavior. It is interesting to note here that the connectivity remains constant until the new node finishes linking to the remaining nodes and only increases again once a new node begins linking. This behavior is not common but needs to be considered. Finally, the connectivity has more range variability than the diameter which more accurately represents the changes in the network than the diameter in this case.

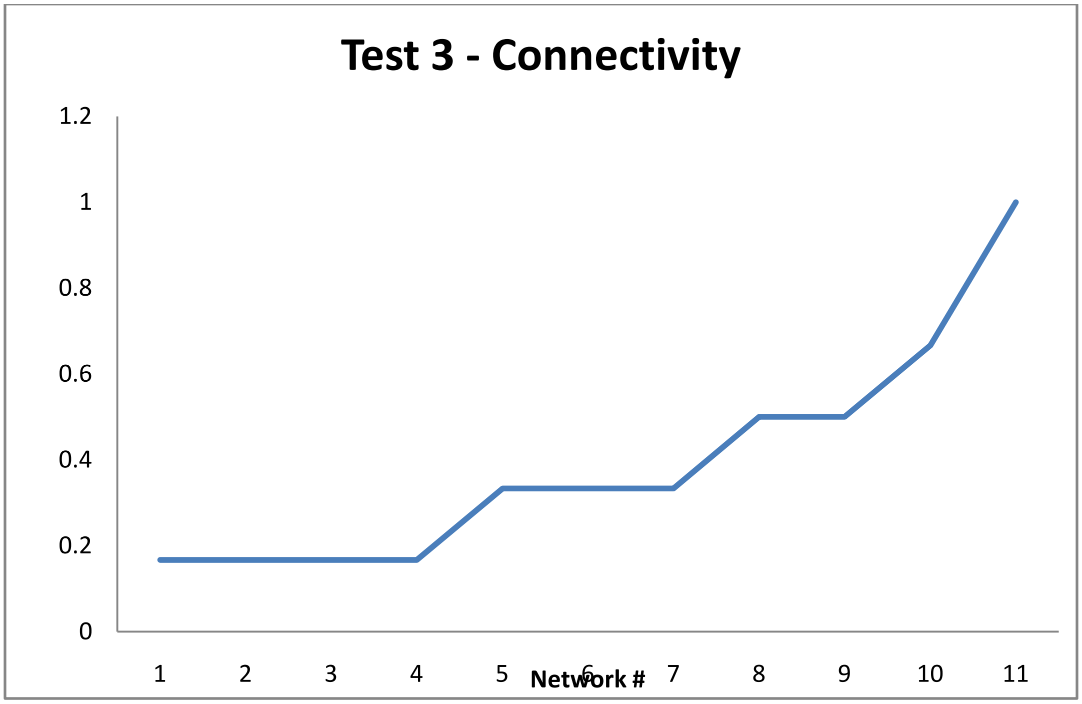


Figure 81 Test 3 Results: Connectivity

The final metric analyzed is the raw Cheeger Constant approximation given below in Figure 82. Unlike the above metrics, the Cheeger Constant has some undesired behavior which can be seen by the non monotonic behavior when an ideal metric would be monotonic. This limits the metric for dynamic use, but the changes are somewhat small and the raw value can be used still in a static sense. This requires some additional considerations which will be discussed in later sections, but it is possible and will occur.

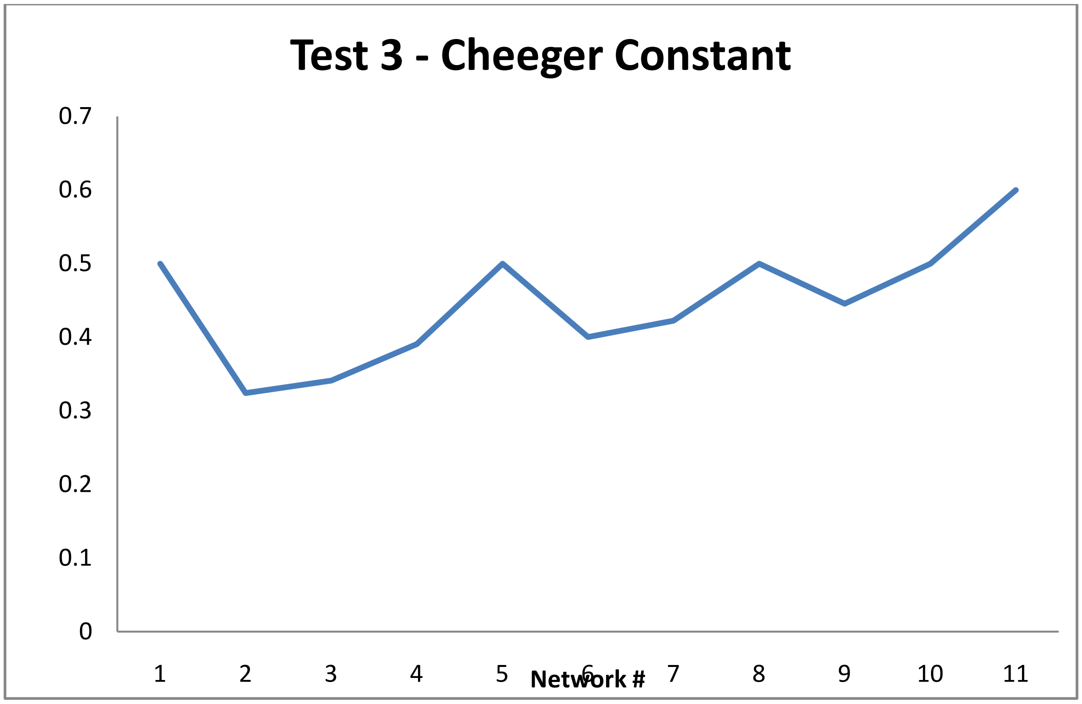


Figure 82 Test 3 Results: Cheeger Constant

With that, the metrics which will be used in a static and dynamic sense have been selected. For the dynamic sense, the metrics will be used to determine in a basic sense what direction the network is moving toward, whether it be more stable or less stable. If the system is becoming less stable, the behavior will be reset to the most conservative case. Less stable changes correspond to any combination of the following: an increase in diameter or a decrease in the normalized connectivity. This concludes the dynamic effect of these metrics on the control, and now the static metrics and their role in control will be discussed. The basic idea of the static control is to transform the given variable values into a control scheme by determining the corresponding control variables. This is shown below in Figure 83.

$$f\begin{pmatrix}\text{Diameter}\\\text{Connectivi ty}\\\text{Cheeger Constant}\end{pmatrix}=\begin{bmatrix}\text{Desired Redundancy}\\\text{Resource Distributi on}\end{bmatrix}$$

Figure 83 Goal of Network Based Control

4.2.3 Determination of Specific Control Scheme

Essentially some way needs to be devised to translate or map the above network metrics into control metrics. Based on the analysis of the network metrics, there is a loose hierarchy in terms of ability to represent network stability given below:

- Diameter
- Connectivity
- Cheeger Constant

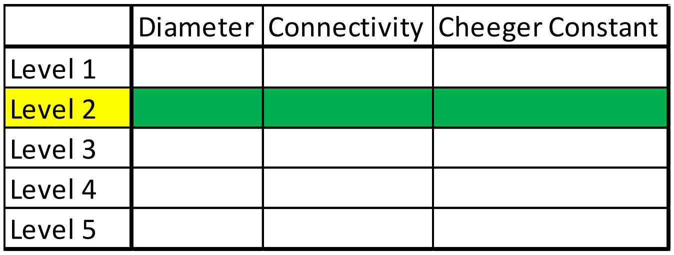
This should somehow be represented in the mapping from network metrics to control metrics. The next thing to consider is the actual means of the mapping. Of the two control variables, the desired redundancy is the most important and it is a discrete variable. This variable is the most important because redundancy is the only way to protect against changes and some forms of disagreement in the system. The resource distribution can take a given redundancy and make it more optimal, but it cannot cause a case that would otherwise fail to not fail. In order to capture these details and to make the mapping scheme as simple as possible a few options or possibilities will initially be chosen. Those are given below:

- The mapping will be discrete in terms of input and output. The two most important metrics (diameter and desired redundancy) are discrete and this will lend itself well to that cause. The metrics will be split into bins from which the control variables will be chosen based on the corresponding bin.
- The diameter should be the driving force in the mapping; the majority of cases should use the diameter to decide the control metrics somewhat independently of the connectivity and Cheeger constant. The exception to this rule will be cases when the diameter does not give information that the system is more unstable than it may appear at which point the values of the Cheeger constant and/or connectivity will be low and drive the system to a more conservative state than it would be in otherwise only depending on the diameter.
    - NOTE: The chosen network metrics are not independent from each other. When the system is fully connected the raw diameter will be one, as will the node normalized connectivity. When the system is disconnected, each of these variables will be undefined (diameter) or zero (connectivity and Cheeger constant). In between these cases one variable might give some

information that the other will not and it is in these cases that having all information will be important.

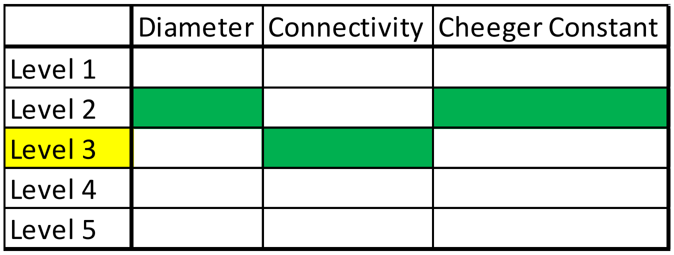
- The bins from which the control variables are given will be based on each network metric independently. That is to say that each network metric will have its own bin. This is due to the above fact that each of the metrics are somewhat dependent on each other and may give similar information in most cases. The worst case of all these bins (in terms of stability) will be chosen as the bin from which to base the control metrics from. This should effectively separate the network metrics while still allowing for the diameter to lead in most cases. Unless the connectivity or Cheeger constant show a more unstable network than what is thought from the diameter, in which case those metrics will be used for the bin selection. A couple examples of this are given below in Figure 84, Figure 85 and Figure 86. In those figures, the bins are given in order from most to least stable, so a bin level 1 represents a more aggressive control scheme than bin level 2. As such the lower bin level will be chosen when there is not agreement between the metrics. The highlighted bin level is that selected for the control variables.
- For extreme enough values of diameter, connectivity and Cheeger constant it may be possible for the system to have agreement, but it will also be difficult for the network to react quickly to any major changes that may occur. In this state the system should choose to be in the most conservative state possible, despite the

fact that pre-consensus has been reached. This will allow the system to translate to a more aggressive control state easily should attractive changes occur, but will not cause the system to over-reach and prevent it from possible failures from over reaching.



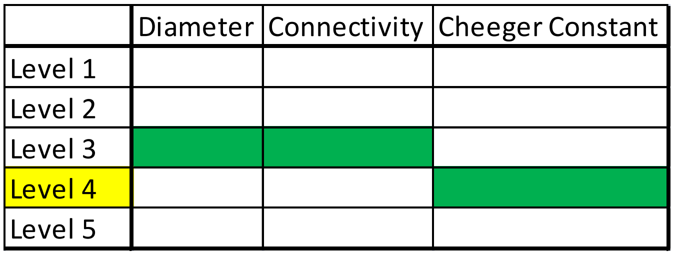
	Diameter	Connectivity	Cheeger Constant
Level 1			
Level 2	🟩	🟩	🟩
Level 3			
Level 4			
Level 5			

Figure 84 Example One of Bin Mapping



	Diameter	Connectivity	Cheeger Constant
Level 1			
Level 2	🟩		🟩
Level 3		🟩	
Level 4			
Level 5			

Figure 85 Example Two of Bin Mapping



	Diameter	Connectivity	Cheeger Constant
Level 1			
Level 2			
Level 3	🟩	🟩	
Level 4			🟩
Level 5			

Figure 86 Example One of Bin Mapping

The actual selection of how to form these bins and what is mapped in terms of control variables will be left for the next chapter. The remainder of this chapter will be devoted to some other issues inherent with the control scheme.

4.2.4 Additional Issues and Considerations in Control Scheme

A few other ideas on how control can be implemented are now discussed. Initially, what can an agent assume each other agent knows beyond its own knowledge? Would it benefit performance to assume that each other agent is reduced in some way and therefore is less able to perform good assignments? The next issue is the use of old information. Is it possible to modify behavior beyond what is given above to still retain some good level of performance even when changes occur? In that same vein is the idea of sacrificing some level of detail in shared information to better ensure agreement? Is it worth it to make that sacrifice to make agreement more likely or easier to reach? The final issue is areas in which control may need to be modified or overridden.

The first consideration involves some means of neutralizing the effect of old information held by the other agents of the system. One possibility is to “over assign” the agent of interest under the assumption that the unknowns of the other agents might give them reduced performance and by taking this into account early, may help prevent failures. The problem with this idea is that while it does help reduce failures, it does so in an unpredictable way. For example, if the desired redundancy is three, each agent can assume that all other agents can only assign two. After the two have been assigned, the agent in question will attempt to fill the gaps. The problem is that this varies somewhat greatly between each agent, and is less effective than just increasing the desired redundancy from the normal control scheme. Each method achieves the same goal, but it is easier to reproduce and implement the desired behavior by doing the latter. This makes sense but leads to an interesting conclusion. Each agent should assume that every other agent knows exactly what it knows and operate under that assumption. That shared knowledge may not be the case; when changes occur the system will move to the more conservative state when it is no longer in pre-consensus. Until that happens the system is in enough agreement to believe that pre-consensus exists. Problems may easily arise when changes occur, but the solution to this problem is to make sure enough redundancy exists to ride out these issues, rather than to make unknown and undeterminable assumptions about other agents and what they know.

This problem is similar to the next issue of the control, in what ways can old information be used? Is it possible under some conditions to operate under the old information even when changes occur? The idea would be to “absorb” a change and continue that current assignment even if otherwise that change would cause a new and different assignment. This idea might be difficult to understand, and therefore an old example will be used to demonstrate this idea. Recall the examples from Chapter 1 based on the cost matrices of Table 8 and Table 13. By continuing with old information, if the changes occurred, the system would continue the same assignment, knowing that it is no longer ideal. This would occur instead of reverting to the baseline assignment as the agent is no longer in a state of pre-consensus. Doing so would give the assignment given below in Table 56, Table 57 and Table 58.

Table 56 Continued Assignment Matrix

0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00
0.00	1.00	0.00	0.00
1.00	0.00	0.00	0.00

Table 57 Raw Task Effectiveness

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.00	0.95	0.00	0.00
0.10	0.00	0.00	0.00

Table 58 Task Effectiveness

0.10	0.95	0.85	0.90

Avg	0.70
Min	0.10

This looks good are first glance, there are no failures and the average task performance is high. When looking at the normalized values of average task performance and minimum task performance (as compared to the assignments of Table 15 and Table 16), the values are .98 and -.19 respectively. The average is quite good, almost as good as the idealized case, however the minimum fails the desired goals of the system: to be at least as good as the baseline case at all times.

One might be able to look at this issue and wonder if it would work for a smaller change: perhaps the only reason this failure of system objectives happened is because the cost change is too dramatic. In fact this is true, if the changed element in the cost matrix is changed to .19 rather than .1, then the assignment would instead be that given below in

Table 59 and Table 60.

Table 59 Modified Raw Task Performance

0.00	0.00	0.00	0.90
0.00	0.00	0.85	0.00
0.00	0.95	0.00	0.00
0.19	0.00	0.00	0.00

Table 60 Modified Task Effectiveness

0.19	0.95	0.85	0.90

Avg	0.72
Min	0.19

This gives a normalized performance of 1.0 and .005. This is much better than before, and in fact passes all desired system requirements. This might lead one to think this idea has some promise, but unfortunately there are major flaws with this idea. If the same example is taken, but the one changed cell now becomes a zero, what happens? The described behavior would be to continue the old assignment but now the assignment is incompatible. This leads to an uncovered tasks and a major failure of the system, which is a major flaw. Once again, this change may be too dramatic and a rational person might suggest that if this is only done in smaller changes might be a good idea. To counter this idea, think of a system in which minor changes occur at each time step. It is easy to imagine such a system where the problem slowly moves into a bad state, but the assignment will never change until pre-consensus happens, and that will never truly happen. For any other changes that occur, agreement might not even be possible; the old assignment might not be able to exist within the new framework because of a difference in numbers of agents or tasks. The idea of being able to absorb some small changes without resetting to the baseline assignment is a good one, but needs to be analyzed more, and will be done so in the next chapter with specific regards to the cost matrix.

One of the main problems that needs to be handled in this research is the lack of agreement that exists between agents. It has been shown over and over again that this disagreement needs to be accounted for in special ways or else the entire control will fail. This can be expected due to the nature of teamwork required in cooperative control. So far the answers presented to this dilemma focus on how to determine and measure the times when agreement is likely and when it is not likely. Another possible way to improve control would be to analyze the amount of agreement required to have a proper assignment without failures and to what limit this agreement can be relaxed. An example of this would be the following meta strategy: If there are four agents each can be assigned to a given percentage of the tasks in the system (IE one and two, two and three, three and four, one and four etc for four tasks) during times without information agreement. The sacrifice in this case is that this assignment will not take into account any information about cost and therefore it is possible for it to even be worse than the baseline case. This may only be true in special instances, but the possibility of an increase in failures from using this strategy exists. This strategy can be modified when addition or removal of tasks occurs somewhat easily, but is more difficult to do when agents are added or removed. In these cases the percentage of tasks chosen must change, and that change must propagate to the rest of the team. The changing of this strategy would have to be coordinated properly over the remaining agents, which is a problem not too dissimilar to that of pre-consensus. For now, these ideas will not be implemented in the basic control context but they leave an interesting avenue for this problem. At the cost of accuracy and complexity of the control scheme, agreement can be gained easier. However without some level of special consideration, this strategy might also lead the system to become more likely to fail as well. This increased likelihood of failure is the main reason this idea is not pursued further.

This concludes the control chapter; the actual tuning of the control method will be discussed in the next chapter.

CHAPTER 5: SYSTEM ISSUES AND NETWORK BASED TUNING

The purpose of this chapter is to discuss the means in which the control technique is tuned. The means of mapping from network metrics to control variables is chosen and modified based on the results of the system. Once again, the overall goal is to minimize if not eliminate the number of failures in the system, which are mainly caused by uncovered tasks. Due to the implementation of the pre-consensus concept, failures can only occur from two causes: removal of agents and changes within the cost matrix (however the other major changes can confound these issues). The cost issue needs to be dealt with in a special way, and will be discussed after discussion of the normal control scheme. The cost matrix on its own *HAS NO EFFECT* on the desired control variables of the system for one simple reason: it is mostly independent from the communication network. In fact, the only relation between the communication network and the cost matrix is the size. The values contained within the cost matrix have no bearing on the communication network or vise versa. A good or stable network has no bearing on whether or not the cost matrix is good or stable and the inverse is also true.

Before the tuning of control mapping is discussed, some other issues must be discussed. One major issue which has not be discussed to this point is the local nature of each agent's knowledge and how that impacts the system. Specifically, each individual agent builds its own idea of the network, and from that network obtains the desirable network metrics. These metrics as well

5.1 Local Considerations of Network Issues and Control

One of the most important features of this sort of cooperative control is that the entire current system knowledge is not available to all agents. The information that is available may be incomplete and will certainly contain old and possibly inaccurate information. When it comes to each agent's knowledge of the overall network this is usually a minor concern. When the network is connected, each agent can build an accurate representation of the network before pre-consensus. However, the use of local information interesting impact when it comes to disconnected networks. From a global standpoint the system is disconnected, but from the standpoint of an individual agent, it is not. Each agent only knows what is communicated to it, so agents that cannot communicate with it essentially do not exist from its standpoint. This can be easily demonstrated with an example. The overall network given below in Figure 87 gives a network made up of two connected components. This network is given a corresponding cost matrix given in Table 61 which will be used to demonstrate some of the features of this disconnected network.

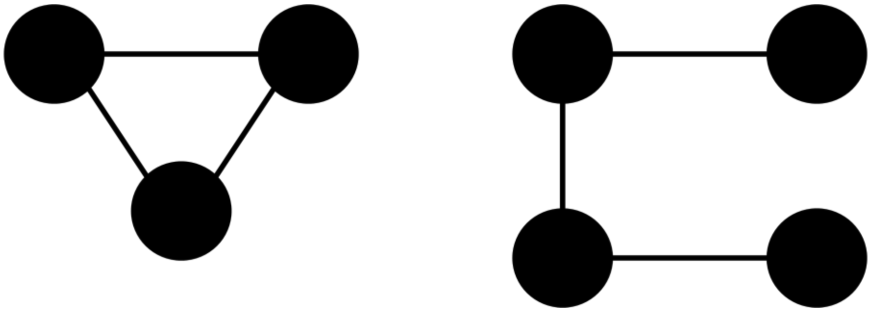


Figure 87 Disconnected Network

Table 61 Whole Cost Matrix for Figure 87

0.79	0.31	0.79	0.49	0.03
0.71	0.59	0.17	0.67	0.53
0.30	0.15	0.40	0.96	0.20
0.90	0.07	0.93	0.06	0.85
0.27	0.46	0.40	0.60	0.68
0.91	0.90	0.55	0.13	0.97
0.99	0.74	0.54	0.93	0.39

The first connected component is given below in Figure 88 with corresponding cost matrix given by Table 62. Despite being smaller, this network is fully connected for the agents within it.

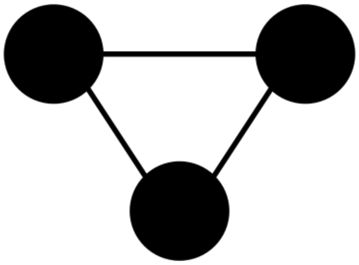


Figure 88 Sub-network 1

Table 62 Sub Cost Matrix for Sub-network 1

0.79	0.31	0.79	0.49	0.03
0.71	0.59	0.17	0.67	0.53
0.30	0.15	0.40	0.96	0.20

The second connected component is given below in Figure 89 with corresponding cost matrix given by Table 63. This sub component is instead sparser and less compact than the other subcomponent.

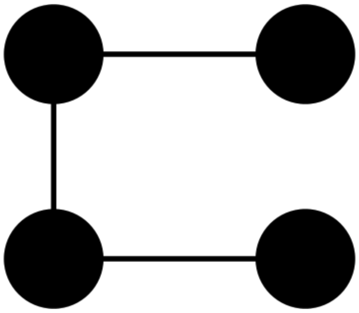


Figure 89 Sub-network 2

Table 63 Sub Cost Matrix for Sub-network 2

0.90	0.07	0.93	0.06	0.85
0.27	0.46	0.40	0.60	0.68
0.91	0.90	0.55	0.13	0.97
0.99	0.74	0.54	0.93	0.39

The interesting note from these two systems comes from the point of view of each agent. Each individual team (connected component) believes itself to be alone and must complete the task as if the other did not exist. This essentially means that the desired redundancy is doubled as two teams are trying to accomplish the mission not knowing that the other exists. This is not the desired behavior, but each individual sub component is smaller and more compact than another network in which the two sub components are lightly linked. This new system would be connected, but also have a much larger diameter than either of the two smaller sub components. Because diameter is important in determining pre-consensus, this larger diameter might not be worth the connectedness. The smaller sub components can react to more rapid changes and reach pre-consensus faster, but are each trying to achieve the overall goal as if the other does not exist. This essentially leads to twice the desired redundancy in the system. This is an interesting fact because while have a disconnected network would seem problematic, it may in fact be better because the smaller sub networks can be more compact than the overall larger network even if it was connected.

5.2 Means of Failures in the Control Scheme

In order to effectively tune the system, it is desired to know when and how failures may occur. When looking at the team of agents, the overall system will always be in one of three states based on the information known and the level of agreement. One state is pre-consensus, where some agreement exists but any change will cause the system to leave this state. Another state is the conservative baseline case that exists when there is no level of agreement of system data between the agents and all agents are in this baseline case. The final case is the difference between the two or what happens when one progresses toward the other. That is when a change occurs and it is known to some agents in the communication network, but not all of them. As it turns out, the only time failures can occur is in this state. These states are discussed below:

- Pre-Consensus State: All agents are in a state of pre-consensus. The system has communicated the information amongst all the possible other agents and enough time has passed without changes occurring so that this state can exist. It is impossible for failures to occur in this state because any change will take part of the system out of this state.
- Baseline case: This is a case of maximum redundancy assignment. Each agent in the system knows of the change which has occurred. The maximum redundancy provides no system failures regardless of what changes occur.
- The other case: This is when part of the system has entered the baseline case in response to a change, but knowledge of this change has not yet reached the entire network so some agents still believe the system is in a state of pre-consensus. Failures occur in this state when the needed change in the control scheme can

only be accomplished by an agent or agents which are not immediately impacted by the change that occurs. This can be seen in both examples in Chapter 1.

These three cases linked together make up the entirety of the system in question. This can be seen below in Figure 90.

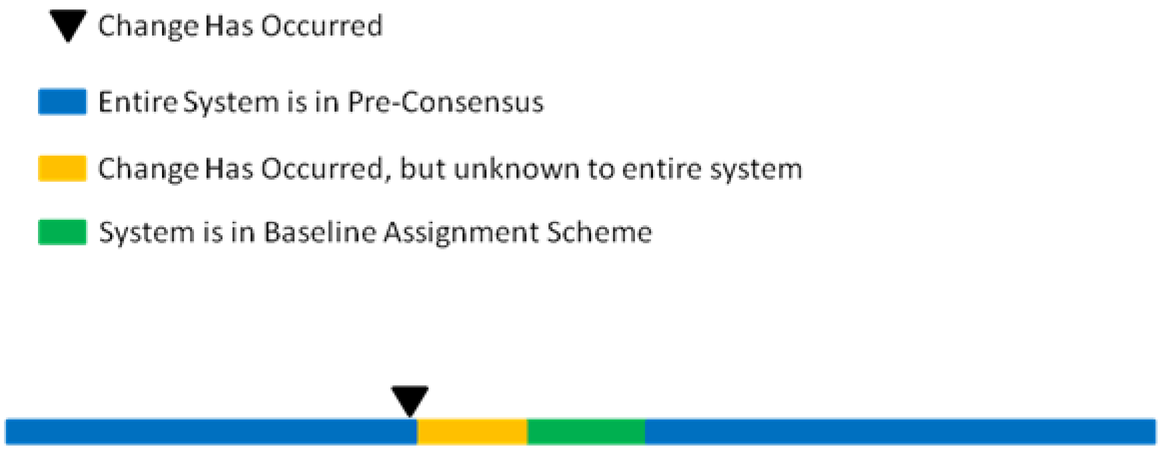

Change Has Occurred
Entire System is in Pre-Consensus
Change Has Occurred, but unknown to entire system
System is in Baseline Assignment Scheme

Figure 90 Illustration of System States

Because the system is only vulnerable immediately after a change, this yields some interesting facts. Firstly, any failure that occurs in the system will only persist for a maximum number of time-steps equal to the diameter of the system after the change occurs. This is equivalent to the amount of steps it will take for this knowledge to spread throughout the system, putting the system in the baseline state and therefore removing the failure. While it is desired that any failure not exist at all, it is good to know that any which appear will only last a finite amount of time, and that time can be small based on the control scheme. For example, by causing any network over diameter three to have maximum redundancy assignment, no failure will last longer than three time steps.

The next result of this effect is that the system is most sensitive to changes which occur at time-scales approximately equal to the network diameter (or a little greater). Any changes which happen more frequently would keep the system from entering pre-consensus at all, and keeping it in the baseline case for the entire simulation. This case is shown below in Figure 91. Any changes which occur less frequently would reduce the percentage of time in the susceptible state. This can be seen in Figure 92. Changes which occur with frequency of diameter of the system are shown below in Figure 93.

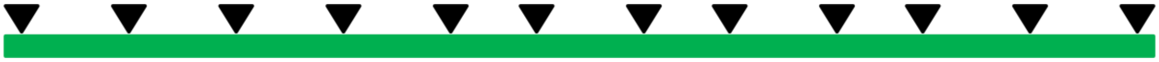


Figure 91 High Frequency Changes in the System

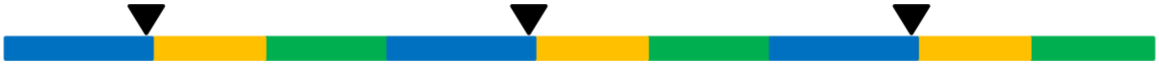


Figure 92 Low Frequency Changes in the system

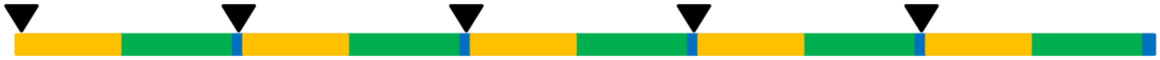


Figure 93 Changes Approximately at Rate of Diameter

As can be seen from these figures, the changes of intermediate frequency leave the system the most vulnerable. This fact should be kept in mind during testing, but may also be taken advantage of when tuning the system. The goal of tuning is to simulate situations in which the system would fail, and changing the control parameters such that the failure can be avoided or at least mitigated. By understanding how to increase the frequency of these failures arbitrarily, this will help to make sure that more failure types will be encountered and accounted for. During normal testing these time scales will not solely be analyzed, as systems with higher and lower frequencies of change will be tested to measure performance over the widest possible range of scenarios. With that the actual means of tuning the control system will be presented in the next section.

5.3 Process of Tuning of Control Parameters

The process used to tune the mapping of control parameters can be summarized in Figure 94 below.

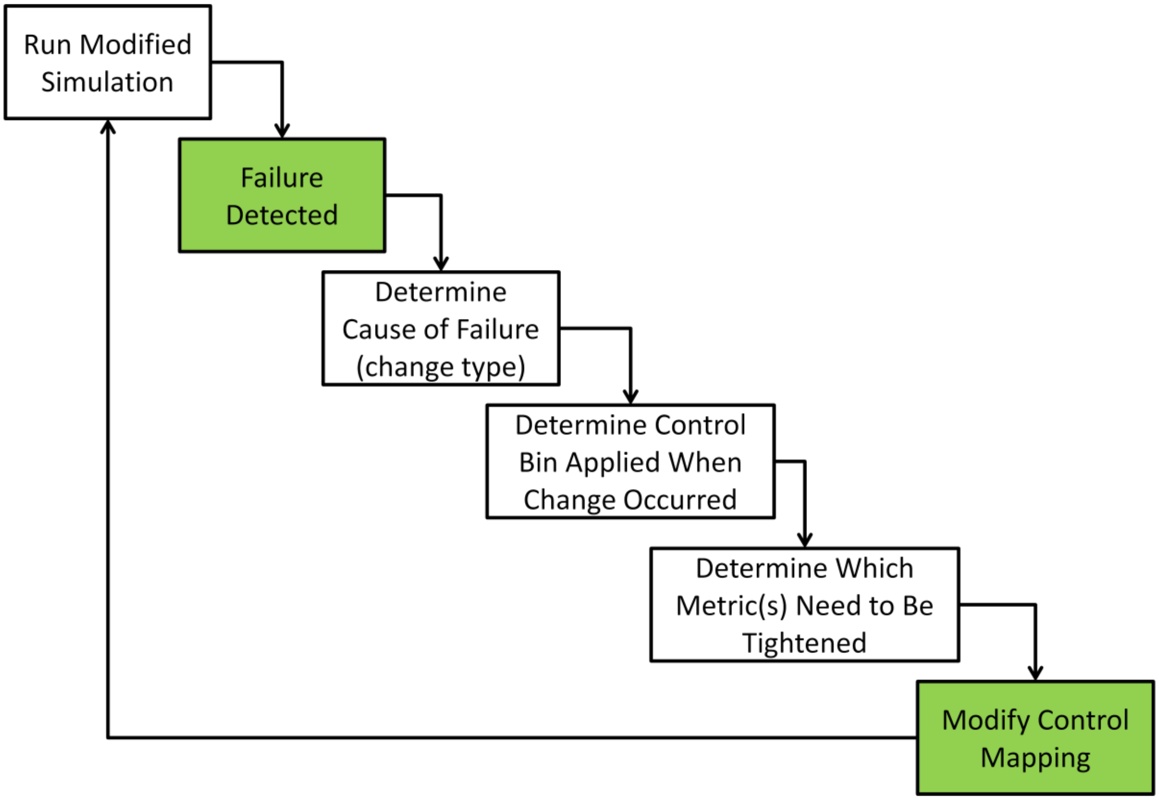

Run Modified Simulation
Failure Detected
Determine Cause of Failure (change type)
Determine Control Bin Applied When Change Occurred
Determine Which Metric(s) Need to Be Tightened
Modify Control Mapping

Figure 94 Control Mapping/Tuning Flow

To start, the simulation is modified such that changes occur at the frequency most likely to bring about failures. This allows for easier simulation of possible failure cases for testing and fewer test runs required. The cases used to test for failures are the same as given in Appendix A with modified change frequencies. Failure is detected when the normalized minimum effective task performance is less than zero or one (total/baseline). This includes failures in which the task is flat out unassigned, and when the assignment simply is outperformed by the baseline case. For the latter, if performance is not at least as good as the baseline case there is no reason to do a modified control scheme beyond the baseline case.

5.3.1 Resource Distribution Variable

From early on in this research, a choice has to be made in terms of performance standards for the developed control scheme. Performance equal to or greater than the baseline case was the desired goal, however the goal could have simply been to prevent uncovered tasks, regardless of performance. This idea is problematic, and has some implications on the resource distribution variable, which will be discussed below.

If only unassigned tasks are considered failures than some tricky assignments can work which are just above no assignment at all. A maximum redundancy can be used, but with varying levels of resource distribution to ensure all tasks are completed, but at different levels. To demonstrate this, consider the example assignment discussed before from Table 17 on. A modified scheme can be made to take the normally unassigned tasks and assign them some small amount of resource. The remaining resource will be applied to the tasks that would normally be assigned, albeit at a reduced rate. The first assignment is made from the initial cost matrix given in Table 8. This assignment and the corresponding raw task performance and raw effective task performance are given below in Table 64, Table 65 and Table 66. This is important to note because implementing this scheme requires that it be done at all times. If it is only done during changes, then it will not prevent any un-assigned tasks.

Table 64 Initial Assignment Pre Change

0.10	0.10	0.10	0.70
0.10	0.10	0.70	0.10
0.10	0.70	0.10	0.10
0.70	0.10	0.10	0.10

Table 65 Corresponding Raw Task Performance

0.01	0.05	0.07	0.63
0.05	0.07	0.60	0.01
0.08	0.67	0.06	0.02
0.64	0.01	0.05	0.07

Table 66 Corresponding Effective Task Performance

0.64	0.67	0.60	0.63

Avg	0.63
Min	0.60

This reduces the overall performance but is still better than the baseline case. When changes occur, this will not be the case. The first change to the system, which corresponds to the cost matrix given in Table 13 is given below in Table 67, Table 68 and Table 69.

Table 67 Modified Assignment Matrix for First Time-Step After Change

0.10	0.10	0.10	0.70
0.10	0.10	0.70	0.10
0.10	0.70	0.10	0.10
0.10	0.10	0.10	0.70

Table 68 Corresponding Raw Task Performance

0.01	0.05	0.07	0.63
0.05	0.07	0.60	0.01
0.08	0.67	0.06	0.02
0.01	0.01	0.05	0.50

Table 69 Corresponding Effective Task Performance

0.08	0.67	0.60	0.63

Avg	0.49
Min	0.08

The next assignment is represented below in Table 70, Table 71 and Table 72.

Table 70 Modified Assignment Matrix for Second Time-Step After Change

0.10	0.10	0.10	0.70
0.10	0.10	0.70	0.10
0.70	0.10	0.10	0.10
0.10	0.10	0.10	0.70

Table 71 Corresponding Raw Task Performance

0.01	0.05	0.07	0.63
0.05	0.07	0.60	0.01
0.53	0.10	0.06	0.02
0.01	0.01	0.05	0.50

Table 72 Corresponding Effective Task Performance

0.53	0.10	0.60	0.63

Avg	0.46
Min	0.10

The assignment for the third time step is represented below in Table 73, Table 74 and Table 75.

Table 73 Modified Assignment Matrix for Third Time-Step After Change

0.10	0.10	0.10	0.70
0.10	0.70	0.10	0.10
0.70	0.10	0.10	0.10
0.10	0.10	0.10	0.70

Table 74 Corresponding Raw Task Performance

0.01	0.05	0.07	0.63
0.05	0.46	0.09	0.01
0.53	0.10	0.06	0.02
0.09	0.01	0.05	0.50

Table 75 Corresponding Effective Task Performance

0.53	0.46	0.09	0.63

Avg	0.42
Min	0.09

The final assignment for this example is represented below in Table 76, Table 77 and Table 78.

Table 76 Modified Assignment Matrix for Fourth Time-Step After Change

0.10	0.10	0.70	0.10
0.10	0.70	0.10	0.10
0.70	0.10	0.10	0.10
0.10	0.10	0.10	0.70

Table 77 Corresponding Raw Task Performance

0.01	0.05	0.49	0.09
0.05	0.46	0.09	0.01
0.53	0.10	0.06	0.02
0.09	0.01	0.05	0.50

Table 78 Corresponding Effective Task Performance

0.53	0.46	0.49	0.50

Avg	0.49
Min	0.46

In each intermediate case the minimum effective task performance is less than that of the baseline assignment case (which is .19). If this is not considered a failure than a simple scheme such as this could work. This is problematic for this research for two reasons. Firstly, this sort of scheme reduces the performance for all times even when it is not needed. Secondly, from a more philosophical standpoint, failures could exist in other forms. Although this problem is somewhat abstract, it could be possible in a real life case that a task performance below a given value is the same as that task not is assigned at all. To prevent this problem, the average and minimum raw task performance should be as high as possible. For the sake of this research, if what is desired is possible then this scheme is superfluous. There should be no reason for this extra caution which causes a reduced performance when pre-consensus is achieved. The only possible reason this case would be considered is that it would prevent all types of unassigned failures from any system change. Whether or not this benefit is worth the reduced performance is a decision that will not be made in this document as it depends on specific applications. Instead, the normal goal of being at least as good as the baseline case at all times will be pursued as it gives the good benefit of improved performance when possible, while avoiding the pitfalls of failure when it is not. Something else to note while on this topic is the role of the distribution variable in the control. With certain values of this, the last assigned task will receive a dramatically reduced amount of resource than it would with a different value of this variable. This can easily make an assignment worse than the baseline case for the minimum effective task performance. The basic premise of this control needs to be re-considered: the system should have enough redundancy to prevent failures, but no more as it will unnecessarily reduce performance. With this issue and the previous one described, the third assignment should be just as strong as the first assignment, as at one point the third assignment will be needed to prevent failures. This implies that the distribution variable should be one for this research.

5.3.2 Desired Redundancy Variable

Now all that is left is to find the desired mapping between the key network metrics and the desired redundancy. The next step in the process is to determine the reasons for why the failure occurred. If cost changes cause the system to fail without any other changes, that will be considered especially in addition to the scheme discussed below and will be further discussed in a later section. This involves looking at the time history of the most recent changes, as well as the control state (or strategy bin) the system was in when these changes occur. With the knowledge of the bins, it is now important to know the value of the corresponding network metrics is determined, which is important in deciding which metrics need to be relaxed in terms of reducing bin size. This process is conducted until there are no failures. Exactly how the bin ranges are changed will be discussed below.

The bins directly represent the desired redundancy which should be used in the assignment process. The minimum possible redundancy should be two, as a redundancy of one would mean that any loss of an agent would cause a failure. Each further bin should have a higher redundancy than the previous until at a certain point the system is unstable enough for anything beyond the baseline assignment to be an unnecessary risk. A sample starting system is given below in Table 79.

.

Table 79 Initial Network Metric Mapping

	Diameter	Connectivity	Cheeger Constant
Level 1	1	0.5	0.3
Level 2	2	0.4	0.25
Level 3	3	0.3	0.2
Level 4	4	0.2	0.15
Level 5	5+		

Each variable will now be discussed. For the diameter the lower the value the better or more stable the network. Therefore the smallest value of the diameter (one) corresponds to the best or highest bin (the most aggressive assignment). Each value of diameter corresponds to a given bin exactly, because the diameter is a discrete variable. For the next variable, the normalized connectivity, the larger the value the more stable the network. This variable is bound between zero and one, so a value of one should represent the most aggressive bin. The table can be read like this: for a given value of connectivity (IE .47), start with the highest level; and see if the value is above the value in the cell. As it is not in this case, move on to the next bin level and do so until a proper bin is found. It the value is below that given in the second to last bin, it will be placed in the last bin. For the example value given above, the bin would be level 2. Cheeger constant behaves in a similar way to the connectivity; with the larger value being more stable. This value is not easily bounded from above, but is bounded from below by zero. The selection of bin operates the same as that for the connectivity.

Once a failure has occurred, the corresponding bins and metric values are known for the system when the change happened. If the problem occurs in a case such as Figure 85 and Figure 86, it is easy to determine that that single metric needs to be changed for that given bin. For example, if the system described in Table 79, in bin level two with a diameter of 1, a connectivity of .41 and a Cheeger constant of .5, the connectivity is the issue. The bin value will be changed to some value slightly higher than .41, say .42. This has the effect of widening the range of the next lowest bin, while translating all other bins. This is graphically represented below in Figure 95. The most aggressive bins are at the bottom; notice that the bin is contracted such that the same change to the modified problem would occur in a different bin.

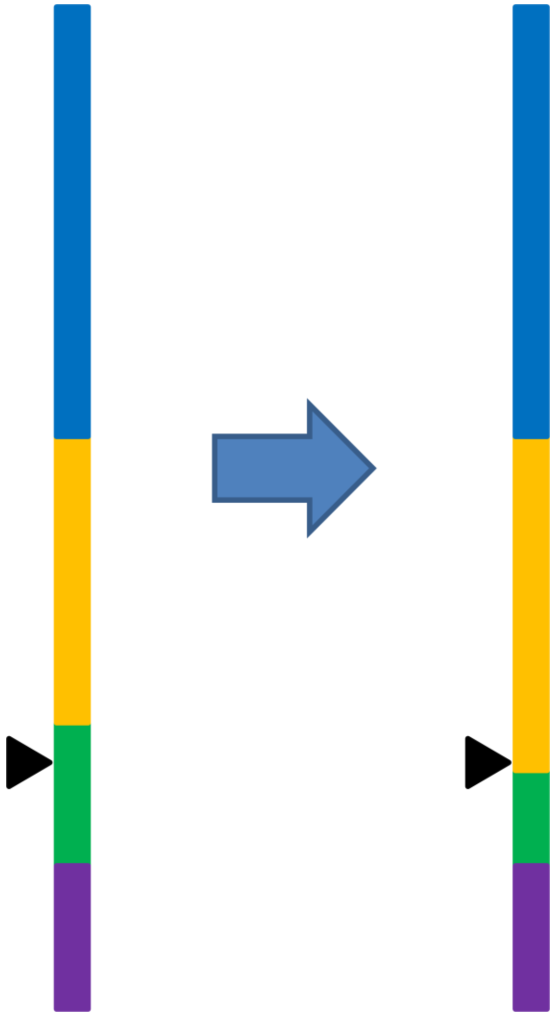


Figure 95 Graphical Example of Bin Modification

There is some ambiguity as to what should be done when a failure occurs in Figure 84. In this case it is not immediately clear which metric bin should be modified. The diameter will be the last variable to be modified as it is a discrete variable. If possible the system should reduce the bin of the connectivity or Cheeger constant instead. The specific metric chosen is based on the distance of that metric to the next boundary.

The closer variable should be chosen as it keeps the system as aggressive as possible. It may be the case later that the other variable is reduced, but this change will keep the system as close as possible to its original state. If both metrics are equidistant to the next boundary, the connectivity is chosen. The connectivity is believed to be a better representation of stability than the Cheeger constant and therefore would be a better representation of why the failure occurs. The final consideration needs to be made is when a bin needs to be removed. For example, if the system bin for connectivity and Cheeger constant has a zero range, then any value of diameter for that bin will not work. In this case, the removed bin is rolled into the next bin as shown below in Figure 96.

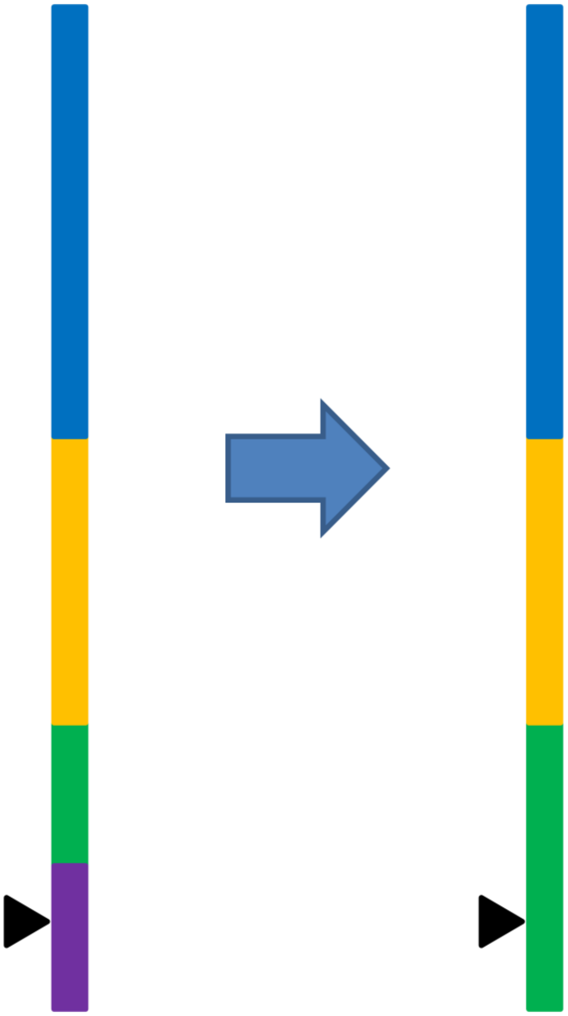


Figure 96 Example of Bin Removal

The final bin selection from this process is given below in Table 80 and Table 81.

Table 80 Network Metrics Bin Mapping

	Diameter	**Connectivity**	**Cheeger Constant**
Bin Level 1	1	0.20	0.20
Bin Level 2	2	0.10	0.15
Bin Level 3	3	0.05	0.10
Bin Level 4	4	0	0

Table 81 Control Metrics Bin Mapping

	Redundancy
Bin Level 1	**2**
Bin Level 2	**3**
Bin Level 3	**4**
Bin Level 4	***Baseline***

This concludes the section on mapping and tuning of the control scheme. In the next section the special considerations made in respect to changes in the cost matrix will be discussed.

5.4 Considerations to Changes in the Cost Matrix

Changes in the cost matrix are different than the other changes in the system in that changes in the cost matrix do nothing to change the stability of the communication network. Another difference in the cost matrix changes compared to the others is that changes in the cost matrix are continuous while the others are discrete. Any smaller change for the other three is still large enough to cause failures, while it stands to reason that a small change to the cost matrix should be able to be absorbed without changing the assignment enough to cause failures if at all. The problem arises in the way to measure which changes are small and which are not.

The desired effect would be to known when the assignment would change based on changes to the cost matrix or to have some sort of sensitivity analysis of the cost matrix. This does exist but it makes the computations required in the control assignment very high. Instead this idea can be implemented in a more simple way. Knowledge of the past assignments exists and can be stored. When a change occurs, the new assignment needs to be calculated and can be compared to the old assignment. It can be assumed that no other agents have knowledge of the cost change, and with that assumption the corresponding assignment matrix can be formed by taking the corresponding row of the new assignment matrix and replacing the row of the old matrix. This process can be seen below in Table 82, Table 83 and Table 84. From here some analysis can be done. Does the desired redundancy decrease below a certain amount for any tasks? Does that assignment change too much in terms of what tasks are assigned? If any of these is yes than the node can be removed from pre-consensus and transition to the baseline assignment case.

Table 82 Initial Assignment

0.50	0.50	0.00	0.00	0.00
0.00	0.00	0.50	0.50	0.00
0.50	0.00	0.00	0.00	0.50
0.00	0.50	0.50	0.00	0.00
0.00	0.00	0.00	0.50	0.50

Table 83 New Assignment

0.00	0.50	0.50	0.00	0.00
0.00	0.00	0.00	0.50	0.50
0.50	0.50	0.00	0.00	0.00
0.00	0.00	0.50	0.50	0.00
0.50	0.00	0.00	0.00	0.50

Table 84 Modified Assignment

0.50	0.50	0.00	0.00	0.00
0.00	0.00	0.00	0.50	0.50
0.50	0.00	0.00	0.00	0.50
0.00	0.50	0.50	0.00	0.00
0.00	0.00	0.00	0.50	0.50

The problem with this practice is that it assumes the changes happen in a vacuum, or that the changes that can be noticed by the given are the only ones that exist. As such this process does not help more than it hurts. Even some small change in the level of redundancy can be absorbed by having redundancy in the system. In addition, the predictive ability is not good. This does not eliminate the chance of failures, but it does decrease performance.

The next idea investigated was to analyze the individual changes in the cost matrix, and reduce the system to the baseline assignment when those changes are high enough. Each agent will analyze the costs that changed (that it knows about) and if any of those costs change by over a certain threshold, will reduce that agents assignment scheme to the baseline case. An example of this is given below in Table 85,

Table 86 and Table 87 with the overall maximum difference for each agent is given in Table 88.

Table 85 Cost Matrix 1

0.119	0.963	0.868	0.926	0.127
0.959	0.473	0.715	0.442	0.570
0.654	0.508	0.702	0.591	0.467
0.735	0.860	0.108	0.308	0.222
0.593	0.155	0.443	0.291	0.782
0.819	0.321	0.688	0.179	0.540
0.887	0.613	0.853	0.706	0.532

Table 86 Cost Matrix 2

0.213	0.947	0.931	0.892	0.100
1.000	0.473	0.837	0.458	0.469
0.679	0.590	0.733	0.473	0.345
0.738	0.956	0.187	0.219	0.217
0.646	0.084	0.485	0.207	0.792
0.936	0.338	0.648	0.278	0.622
0.950	0.683	0.791	0.587	0.619

Table 87 Difference between Cost Matrix 1 and 2

0.094	0.016	0.062	0.034	0.027
0.041	0.000	0.122	0.016	0.101
0.025	0.082	0.031	0.118	0.122
0.003	0.096	0.079	0.090	0.005
0.053	0.070	0.042	0.084	0.010
0.117	0.017	0.039	0.098	0.082
0.063	0.071	0.062	0.119	0.088

Table 88 Maximum Cost Change with respect to Each Agent

0.094
0.122
0.122
0.096
0.084
0.117
0.119

If the baseline threshold is .1 then the highlighted cells in Table 87 and Table 88 are those which are over that threshold and will reduce those agents to the baseline assignment. This idea has some problems; the main one is that it does not eliminate failures from this change. Like the addition and removal of agents, this problem cannot be eliminated unless the threshold is set to zero. The smaller the threshold the less likely it is for failures to exist; in some small circumstances this can still happen. The other problem is that this considers each element in the cost matrix as equal when they are not. Due to the nature of the optimization, the higher values in the cost matrix are more likely to be selected for assignment than the lower values. Therefore it stands to reason that only changes to these high valued cells, or changes which make an otherwise low valued cell into a high valued cell are those that matter. However exploring this idea did not do well in greatly reducing the amount of failures, and it became difficult to know which cells were going to be important in the assignment without knowing how the assignment would operate and the information contained by the other agents. This idea was abandoned, and instead each cell was treated as the same.

With this idea in mind, the next step was to choose the proper threshold. Test cases were run with only cost changes and the results are given below in Figure 16 and Figure 17.

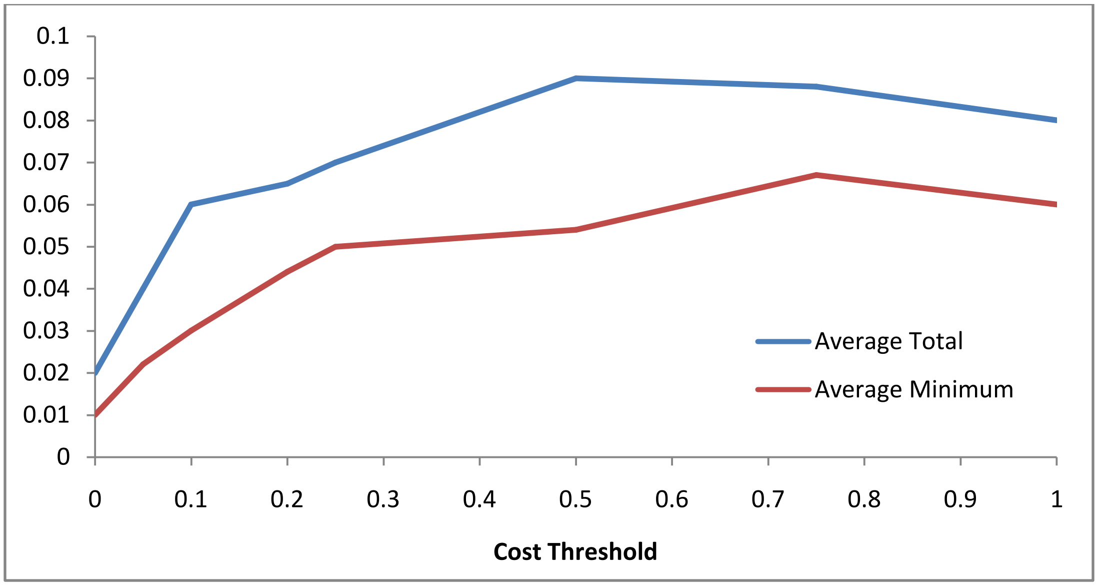


Figure 97 Effective Task Performance vs Cost Threshold

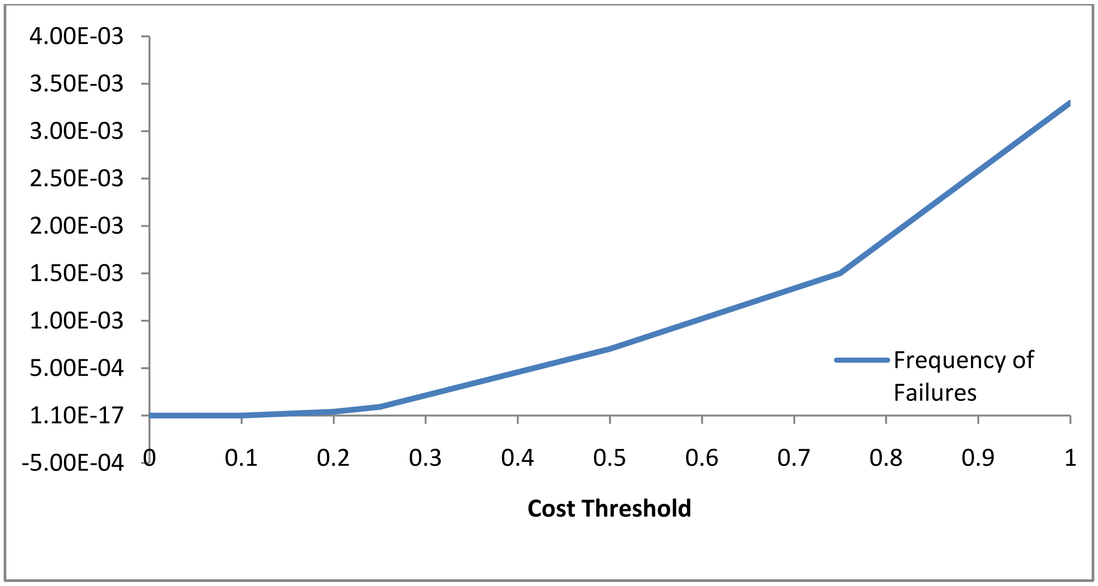


Figure 98 Frequency of Uncovered Tasks vs. Cost Threshold

From these figures, it becomes clear that the best value which still eliminates failures is .1. These plots also give an interesting insight into the behavior of this system, specifically Figure 97. At high values of cost threshold, the amount of failures that occur reduce performance despite the fact that no otherwise good cost matrices are considered bad and behavior reduced to the baseline assignment case. For lower values of cost threshold, the amount of failures is mitigated, but to do so a great number of otherwise OK changes which occur must be thrown out. This leads to the fact that for some non-zero amount of failures, the performance can be maximized. This occurs because there is balance between the aggressiveness of the cost threshold while still allowing many of the otherwise good changes to remain without resetting of assignment to the baseline case. Even though it is not the goal of this research, it is interesting to note that if the performance metrics are desired to be maximized that occurs with some non-zero value of failures. That is to say that until some point, the reduction of failures helps the overall performance, and any movement beyond that point reduces failures by considering many changes "bad" because of the small likelihood that they are in fact problematic.

At this point it is noted that for a different system objective, a different tuning scheme can be used to best meet this different goal. For this research, it is desired that system failures be minimized, however another common goal is to maximize performance. These are exclusive goals in this case as demonstrated by Figure 97 and Figure 98 (cost failures act similarly in this way to failures resulting in removal of agents); maximum performance does not occur at the same value as minimal failures. If it is desired that the system have maximum performance, which requires accepting some level of system failures, the tuning scheme needs to be changed. Rather than changing the bin size for any failure, the complete test must be run and bin sizes modified in order to achieve the desired amount of failures. This could be done with a genetic algorithm using the size of each bin as a variable. In addition, more focused problem types would allow for a more focused test bed, and a new tuning could be found to work better for this case. This can occur regardless of the overall objective.

With the use of network variables to dictate the control of the system (both in reaching pre-consensus, and in determination of control variables while in pre-consensus), research question one has been answered. That research question is given below.

1. Can a metric or set of metrics be found which accurately represent the dynamic effects and changes of the system as it pertains to Cooperative Control? Can these metric(s) be used as a means to trigger changes in the control scheme?

The answer to this question is yes; the static variables of diameter, connectivity and Cheeger constant and dynamic variables of diameter change and connectivity change have been used to dictate the cooperative control of the system. The given control mapping from above has been tuned to effectively prevent failures in the system for a test case that was designed to cause failures. This fact should indicate a lack of failures when this control mapping is applied to the unchanged test bed.

This concludes the discussion of tuning of key parameters in the cooperative control of this system.

CHAPTER 6: RESULTS AND DISCUSSION

The purpose of this chapter is to show the results of the main control testing. This captures the complete proposed research and results accordingly. To save space in the charts and figures of this chapter, no failures were detected unless otherwise noted.

6.1 Standard Test of Control Performance

The main goal of the control is to create a cooperative control scheme which can adapt based on the communication network to take advantage of different requirements. In some instances the system may be stable and more aggressive control schemes may be possible, but in other cases more conservative schemes will be needed due to the lack of any sort of agreement of the agents. In order to test the effectiveness of the control under a variety of network schemes and other combinations, a test was created which will simulate many different problem types. This primary test needs to contain very demanding changes, both in impact and frequency as these cases are the most likely to cause failures in the system. A reduced size Monte-Carlo analysis was created for the major inputs in the system and those can be found in Appendix A. This simulates different network types, change types, impacts, frequency and other changes.

The results of this case can be seen below in Table 89.

Table 89 Results of Main Test Case

	Average Total	Average Minimum
N_T	0.03	0.02
N_B	1.16	1.096

As said before, this test case is very demanding so it is not surprising that the results are relatively low and close to the baseline case. Despite this difficulty, the use of the baseline normalization can be seen. Despite the presence of impactful and frequent changes; performance is increased by almost ten percent when compared solely to the baseline performance. One note for the baseline normalization is that in most cases the performance of the minimum task is usually lower than the performance of all tasks averaged. Baseline normalization also gives an alternate meaning via its reciprocal; the amount of agent resource required to create behavior equal to the baseline case which is accomplished via cooperation. By taking the lower value of total and minimum task performance, then taking the reciprocal of the corresponding value, this gives the resource which could be used by each agent to give the same performance as the baseline. For example, using Table 89, the lower of the two values is for the minimum, which is 1.096. The reciprocal of this value is .912. This means that agents with a maximum resource of .912 using the above control scheme would have equal overall performance to agents with a resource of one which do not communication or cooperate with each other.

As discussed in previous sections, cases of high frequency/ high impact changes will likely cause the system to never enter pre-consensus or do so only. There is nothing that can be done about this as entering pre-consensus prematurely or in error will more than likely cause more harm than good. In addition, cases in which pre-consensus is possible may never enter a state of more aggressive assignment because the system is not in a stable enough state to be able to do so. In short there are many reasons why the system might have lower performance than expected, especially in such demanding circumstances. The good part of these results is that no failures occurred, and the results did exceed those of the baseline case in some cases. This is in direct relation to the second research question of this study, presented below.

2. Can a control scheme be developed which will consistently perform no worse than the baseline scheme of no communication?

As the main test case was designed to be the most taxing and strenuous possible, passing this test answers the research question with a yes. The only issue is that failures cannot be absolutely said to be eliminated, but for all practical purposes the chance of failure is small enough to not be impactful. In addition, any failures which may occur will be finite in length and bounded above by the diameter of the system.

At this point it needs to be mentioned that the assumptions made at the beginning of the research directly contribute to the lack of absolute assurance of failure prevention. The goal of the initial assumptions were to make the system as difficult or problematic as possible to analyze the limitations of control and see if it was even possible in this case. The idea was that if a cooperative control scheme was successful for the most restrictive case, that scheme would also work for less restrictive cases with equal if not better performance. Even in that most restrictive case, two of the four major system changes can be accounted for with an assurance of zero failures, regardless of the frequency or measure of impact of those changes. These changes are addition and removal of tasks and link modification of the communication network. The changes in the cost matrix are a special case; they can be accounted for with an assurance of zero failures, but at a significant cost of system performance. This research discussed specialized methods to effectively give, but not absolutely promise, zero failures while still maintaining an increased performance. It is the author's view that the specialized methods are worth the lack of assurance with respect to system failures, as the chances of failures are extremely small. The only change that remains problematic is the addition and removal of agents in the system. The main concern with this is that the other agents in the system will not realize the agent is missing until at least one time step has passed. During that time the missing agent's assignments cannot be replaced. If enough agents are removed and the redundancy is not enough to absorb this impact then failures will occur. A few assumptions can be relaxed which will allow for cooperative control schemes assuring zero failures. Those are given below with brief explanation of how that promise can be kept.

- Knowledge of when an agent is removed before the next assignment.
    - Just knowing that an agent is gone is enough to revert the assignment scheme to the baseline case for the entire system. This knowledge does not necessarily mean that the system is fully connected; the rest of the information can travel in the normal channels.
- Communication occurring after change but before assignment.
    - This would allow knowledge of an agent's removal to its old neighbors before the next assignment round. By creating a specialized scheme where at least one of those neighboring agents has the able to assign each of the tasks, then the agent's removal will alert an agent who can fill its role with enough time to prevent failures.

Other assumptions may be relaxed to give a similar effect, but the above assumptions are the most minor and easiest in terms of changes in the cooperative control scheme. These changes would be incredibly beneficial to the system as it would allow for a much lower level of redundancy which would greatly improve performance.

6.2 Less Stringent Testing and Comparison to Static Control Schemes

This section will discuss a few cases in which the test-bed was relaxed to look at less stringent situations which can better show the control performance. If pre-consensus can be reached in more test case, it will give a better example of the possibility of improved performance via the control scheme formulated by this research. It may be unlikely that the sort of changes and impacts that are allowed to occur in the standard test will occur as often. The changes analyzed via the standard test case can be relaxed in frequency to analyze this. It is also unlikely that the impacts will be as high as allowed in a single time-step. In the normal test bed almost the entirety of the tasks and agents can be removed or replaced in a single time step, the entire communication network can change in single time step, and the entire cost matrix can change dramatically. Limiting these changes somewhat, as well as their frequency will still allow drastic changes to occur, but they will be more gradual, allowing the system to respond quicker than in the nominal test case. As the nominal case had effectively zero failures, these other cases should also have zero failures, but their performance should also be increased dramatically. The results of this analysis are given below in Figure 99 and Figure 100. The test bed for the medium and low stringency cases are given in Appendix B.

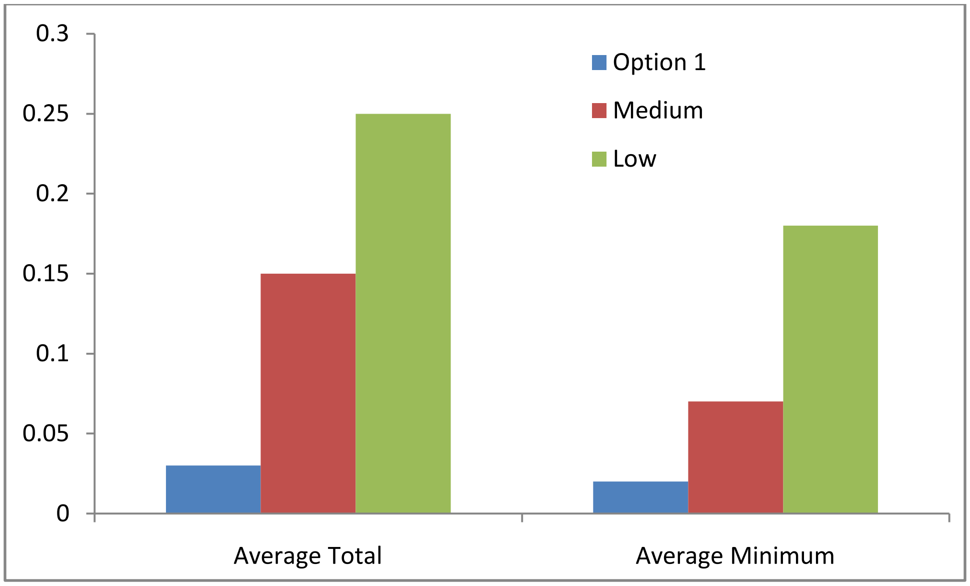


Figure 99 High (Nominal), Medium and Low Stringency Testing (Total Normalization)

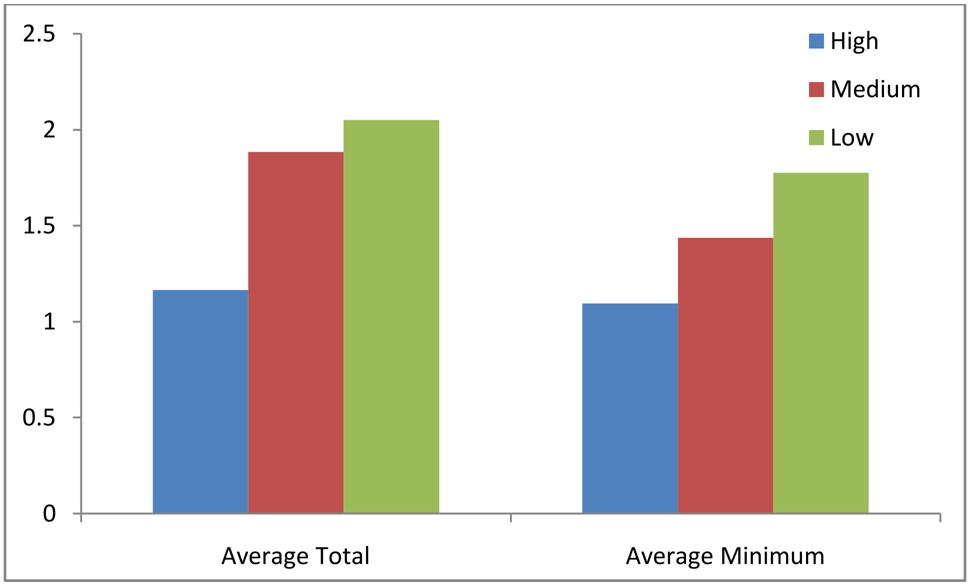


Figure 100 High (Nominal), Medium and Low Stringency Testing (Baseline Normalization)

As expected, the results are greatly improved, while the system is still allowed to have a great amount of change, albeit over time rather than immediately. Implementing these more gradual changes shows the quality of impact of the control scheme via the baseline normalize values. In the medium stringency case, performance increases by approximately fifty percent, and in the low stringency case, performance increases by almost one hundred percent. . Results could also be better if the control mapping was tuned specifically to these cases as they are less demanding than the standard case. For these results, the less stringent cases were conducted using the same control mapping from the nominal test case. By re-formulating the control mapping for the less stringent cases, performance can only improve.

The next case to consider will compare the standard test cooperative control to static cases considering set levels of redundancy. For the standard test, the amount of failures would be incredibly high with static testing, so different testing means were developed. Each case starts with five agents and tasks, with the agents connected in a ring network. Each change is considered separately, with every timescale for change tested. This gives twenty four tests, with each change having six timescales. Two cases were chosen, one for average performance amongst cases for the static control methods, and one for the worst performance among the static control methods. These demonstrate essentially a case in which static cases perform well, and perform poorly. Both will be used as comparison to the developed control to demonstrate the effectiveness of the adaptive nature of the control. The good case is given below in Figure 101, with the bad case given below in Figure 102 for the total normalization. These two cases are also given for the baseline normalization below in Figure 103 and Figure 104.

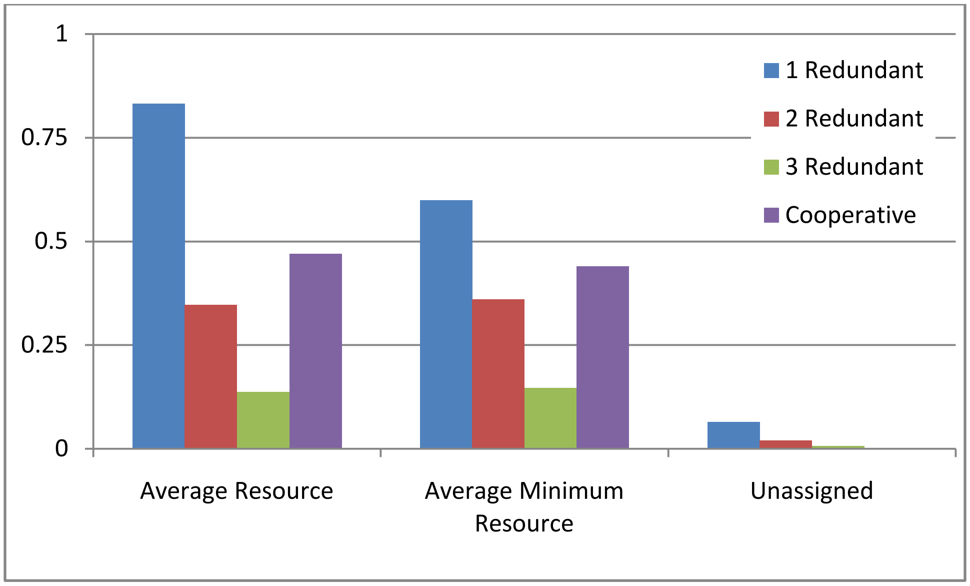


Figure 101 Comparison of Static Control to Cooperative Method "Good" Case (Total Normalization)

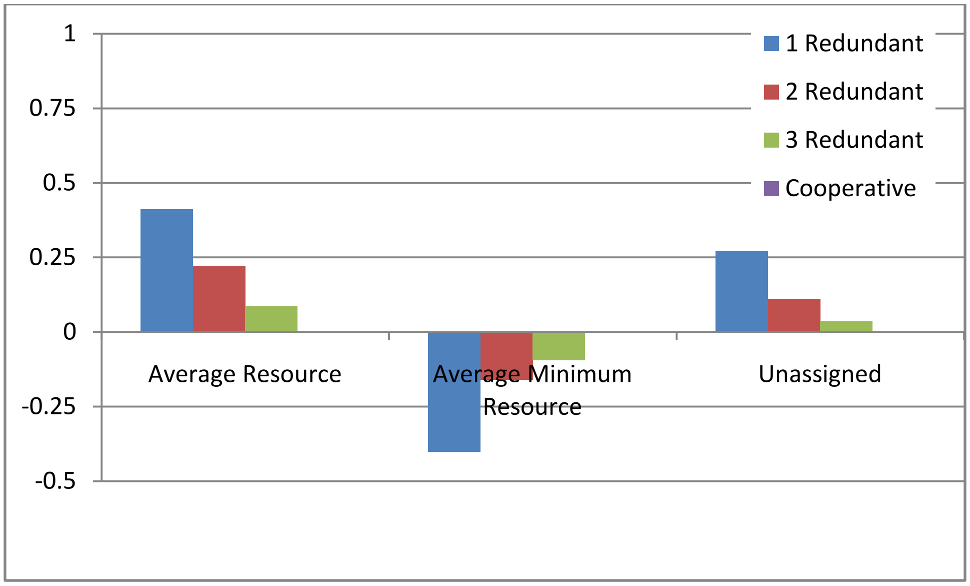


Figure 102 Comparison of Static Control to Cooperative Method "Bad" Case (Total Normalization)

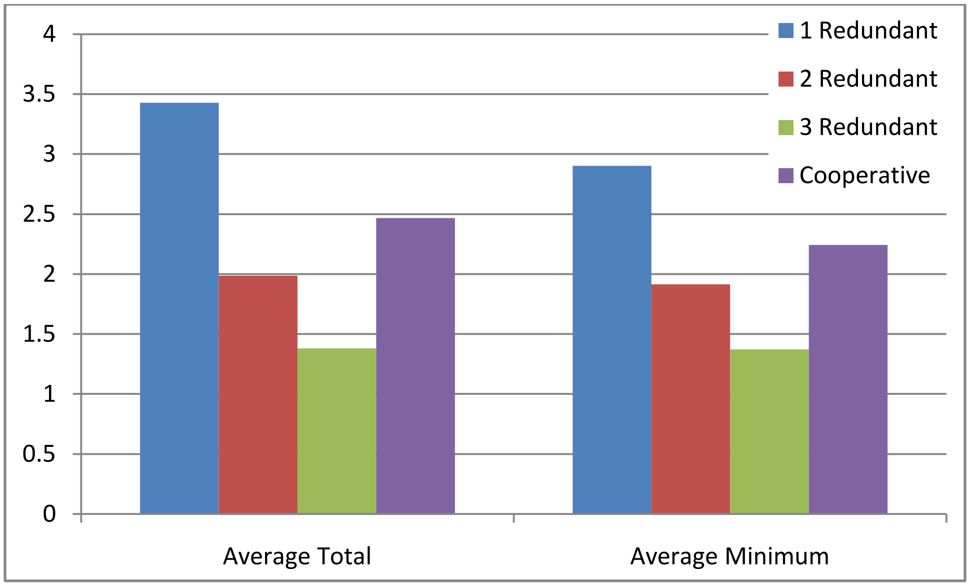


Figure 103 Comparison of Static Control to Cooperative Method "Good" Case (Total Normalization)

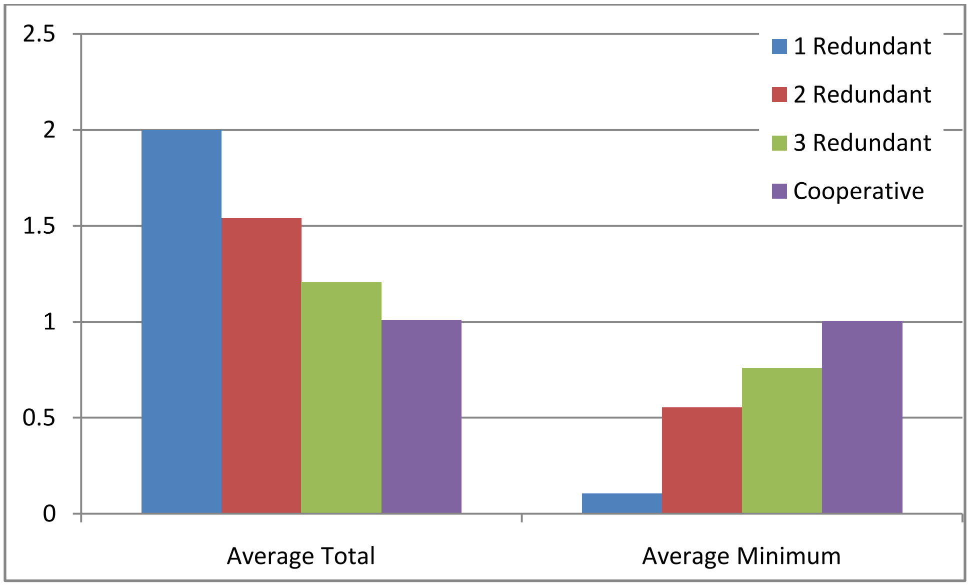


Figure 104 Comparison of Static Control to Cooperative Method "Bad" Case (Baseline Normalization)

What is important to take from these figures is that the cooperative control scheme described in this research can capture the good of almost the most aggressive static schemes when it is possible, while also mitigating failures and still maintaining some improved performance in the bad cases. Even in the “good” case, the highest level of redundancy still does not prevent unassigned tasks and does so at reduced performance. The sacrifice of this flexibility and the prevention of failures is that the performance in good cases will still not be as good as the most aggressive techniques. This is the cost of doing business and allowing the cooperative control technique to adapt while still preventing failures. Note that via the baseline normalization from Figure 104, the minimum task performance is lower than one in all of the static cases, showing that to achieve the same average performance as the baseline; resource availability would actually need to increase. The reason for this is the non-zero failure rate from Figure 102 (which is as high as .25) and even in Figure 101, showing that even in the good case some non-zero level of failures exists.

The final note to be taken from this chapter is that the designed cooperative control scheme meets the desired research objectives. The system is at least as good as the baseline case of no communication; and performance is much better as a result of cooperation. Even in the highly demanding standard case, performance increases by almost ten percent when compared to the baseline case. By analyzing less stringent cases (which still allow a high amount of change, only make it more gradual rather than sudden) performance increases by fifty to one hundred percent when compared to the baseline case. The comparison to static techniques also shows the need for the adaptive scheme to transition between different problem situations, while also showing that the increase in performance must come with a detailed control mechanism to alleviate the issues of change and disagreement in such a variable system. This addresses the second research question and verifies the associated hypothesis.

CHAPTER 7: SYSTEM WEAKNESSESS AND EXPLORATION OF POSSIBLE SOLUTIONS

The purpose of this chapter is to analyze weaknesses of the control scheme and possible problems which may arise by removing or lessoning of previous assumptions. The main weaknesses detected in the results section deal with agreement and achieving pre-consensus in difficult situations. The assumptions of the system are reproduced below.

- Each enemy must be linked to at least one ally
    - Otherwise all cases including baseline will fail
- Each task must be assigned by at least one agent for every time step
- Changes may be as dramatic as going from the maximum to minimum number of allies or minimum to maximum number of enemies in one step
- Communication: No errors beyond those given already in the system (which may simulate false positives, losing communication etc)

The additional cases studied will each relax some of the assumptions and analyze modification (if any) which must be made to handle these situations. From there the results of these cases will be analyzed. Those extra cases are given below, with the assumptions that are relaxed.

- Periodic Task Assignment
    - This case considers task which must be assigned every X time steps rather than every single time step. This will test the method in situations when communication may occur more rapidly than system action as well as non-coordinated tasks.
- Error Analysis
    - This case will assume that errors exist in sensor, transmitter and receiver, and look at how these errors impact the performance of the system. Even in pre-consensus, some disagreement will exist at all times due to this error.
- Small Team Analysis
    - This case will look at situations in which the number of tasks and agents greatly exceeds the assumed maximums. Due to its large size, the system is less likely to be able to reach pre-consensus, and other avenues are analyzed.

7.1 Additional Testing of Special Cases

The purpose of this section is to analyze cases which stretch the assumptions about the system and explore additional issues which may arise in real life systems.

These cases will be presented in order of “difficulty” in terms of what issues arise and how difficult these issues are to solve or account for. This includes the difficulty in terms of the modifications to the cooperative control scheme as well as the added strain to the system.

7.1.1 Periodic Coverage Analysis

This case will consider a slight modification to tasks and the required assignments for them. In the standard case, each task needs to be assigned for every time-step. For this modification, that will be relaxed, and some percentage of the tasks will only need to be assigned ever X time-steps. It is assumed that the value x will not change during the simulation; once a task has been introduced with a given value, it will remain constant. This can be the case in certain problems such as surveillance when some targets need to be refreshed less often to get an idea of what is going on. Another way to think of this problem is that communication can occur multiple times between assignments (if all tasks are periodic). This case is somewhat easy to implement as it is a less intensive version of the original. The original cooperative control scheme could be used in this case and no additional failures would occur. In fact, the likelihood of failures would decrease because the task which is failed might be one that does not need to be assigned at each time step and can be picked up later. However this does not take advantage of the fact that the agents can better spend their resources by ignoring the periodic tasks when it is not necessary to assign them. Tasks that do not need to be assigned will be removed from the cost matrix during assignment and will not even be considered.

Three additional variables need to be added to the system to accommodate this change, and those are given below:

- Percentage of tasks which are periodic: this dictates the number of tasks which need to be assigned every X time steps rather than every single time step. When new tasks are added, they are checked vs. this chance to see if they are periodic.
- Maximum and minimum values of periodicity: For tasks that are deemed to be periodic, these variables set the bounds on that periodicity (or the value X). For this simulation the value of the periodicity is chosen at random without preference amongst every integer value within this range.

The next issue which needs to be considered in this case is how to calculate the performance metrics for the system for periodic assignments. One option is that each periodic task's performance is only calculated every X time steps. For each time step between this calculated and the last, the time history of the various assignments is stored. The best assignment amongst all the time steps is chosen as the representative assignment for this task during this period. It is this value which is used to calculate the standard performance metrics. This can be seen below in Table 90. The yellow highlighted cells represent the effective task performance for each time-step while the green highlighted cell represents

Table 90 Example One of Performance Calculation

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Time Step 1	0.00	0.00	0.00	0.00	0.50
Time Step 2	0.00	0.00	0.00	0.60	0.00
Time Step 3	0.00	0.00	0.70	0.00	0.00
Time Step 4	0.00	0.90	0.00	0.00	0.00
Time Step 5	0.80	0.00	0.00	0.00	0.00

The other option to consider is that the specific time-step in which the assignment is made matters. That is to say that for a task with five periodicity assignments made any amount of time steps less than five from the previous step is not beneficial. This case is much more restrictive as assignments made between these time steps essentially count for nothing. This is a more interesting and restrictive case because it rewards some amount of synchronization of the agents in the system which may or may not be possible due to communication limitations. This idea can be demonstrated by considering the same assignment schedule as Table 90, but with the new rules. This is given below in Table 91.

Table 91 Example Two of Performance Calculation

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Time Step 1	0.00	0.00	0.00	0.00	0.50
Time Step 2	0.00	0.00	0.00	0.60	0.00
Time Step 3	0.00	0.00	0.70	0.00	0.00
Time Step 4	0.00	0.90	0.00	0.00	0.00

Time Step 5	0.80	0.00	0.00	0.00	0.00

The main difference in these cases is the punishment for mistiming. In the first case the assignments made in different time steps essentially act as redundancy in the system. In the second case it does not even count as that, essentially these assignment are only wasted. Another consideration can be made to count some values of the second case for failures. In such a case, if the desired time step for assignment is missed or failed, the system can consider the most previous assignment (even if it is not on the desired step) and reformulated the desired time step based on this. If the final row in Table 91 is all zeros, then the most recent assignment (Time Step 4) will act as the start of a new scale, and the next desired Time Step for assignment will be Time Step Nine (5+4). This can be considered option three. This option is in between one and two in terms of limitations. Assignments not made on the desired step are not completely wasted as they are in option two, but they do not act as normal redundancy as they are in option one. Each option will be considered.

One key addition in terms of control technique is some way to synchronize when changes occur. If the last time an agent has assigned a given task is included in the information communicated, this can easily be accomplished. If a new agent is added to the system, it will not have any knowledge of the desired time step each task should be assigned. Until it receives the information of when each task was last assigned, the agent must go into the baseline assignment scheme. One this information is received the agent can enter the normal assignment scheme. What this means exactly is different for each option considered above, however any synchronization is not possible in the baseline case. This is because the baseline case is the same as having no communication, and synchronization is not possible without communication.

For option one, the system acts almost exactly like the system would in the normal case. While assignments do not need to be made every time step for periodic tasks, there is no penalty for which time step assignment is made during the proper range. For the overall control performance, there is not much difference from the normal step. While the baseline case cannot be synchronized, the effect is still that of having normal redundancy, and the lack of synchronization does not interfere with the baseline cases ability to assign an X periodicity task every X time steps.

Option two is a little different in terms of what the baseline case means. Any new agent added will have no idea what the desired time step is for assignment of each periodic task, and therefore will have no choice but to ignore the periodic effects and assign those tasks every time step. To do otherwise could result in a failure from an unassigned task.

The baseline case for option three is more like that for option one, but it is not as advantageous as it was in assignment one. Any times the desired time step is unassigned for a given task, and then the previous assignment will be used. Each agent will be able to assign these tasks every X time steps and take advantage of being able to ignore some periodic tasks without worry of failure.

Considering each option additional tests were run with a .50 chance of periodic tasks and each task having periodicity between three and five, the remaining inputs being the same as the standard test case given in Appendix A. The results of these tests are given below in Table 92 and Figure 105.

Table 92 Results of Periodic Options

	Average Total	Average Minimum
Option 1	0.03	0.02
Option 2	0.25	0.18
Option 3	0.14	0.10

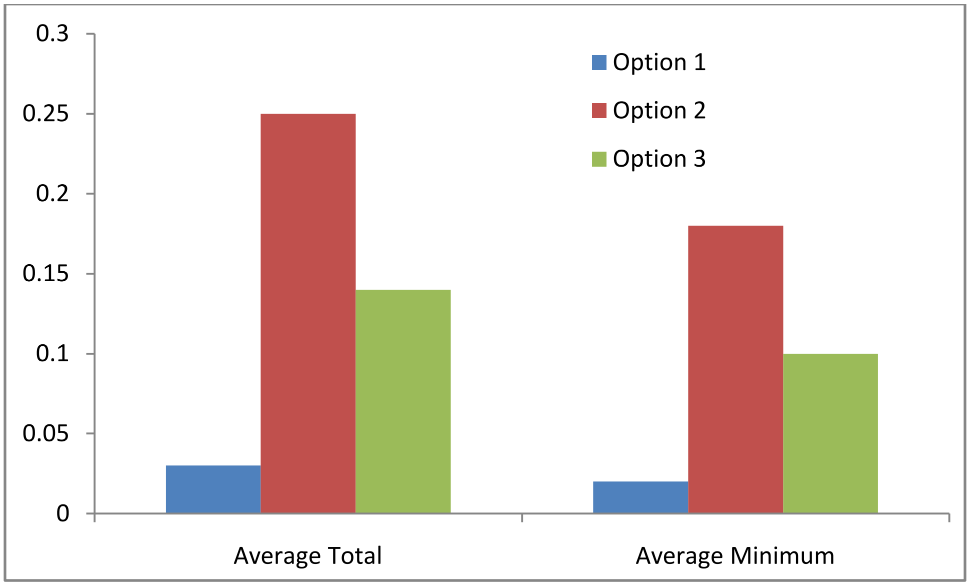


Figure 105 Results of Periodic Option

While the results increase, it is difficult to determine which part of the performance increase is due to the less taxing assignments vs. the added synchronization of the cooperative control case (and reduced performance of the baseline case). Between options two and three, option two has increased performance due to the high demand on the baseline case.

The final consideration in this problem consideration will analyze possible ways to reduce failures while also increasing performance by thinking of failures and assignments for periodic tasks as redundant systems in parallel. The main reason why the control mapping is so strict is to minimize failures as best as possible. For the sake of an example, assume that the standard control scheme without periodic tasks is one failure per every 10,000 tasks. If that same scheme is applied to a periodic agent of periodicity two, then the chance of failure for that task has now been reduced to one failure per every 100,000,000 because now a failure in the periodic case would require two consecutive failures in the old case. Instead assume that the two periodicity case should have the same failure rate as the one periodicity case, of one failure in every 100,000 tasks. This would lead to a normal assignment scheme for this case with a failure of one in every 100 tasks. Such a control scheme could be much more aggressive and not increase failures at all, due to having a redundant assignment. One way in which this could be implemented is to have a much more aggressive control scheme apply assignments at twice the desired rate at much more aggressive rate in terms of desired redundancy. As it stands this research did not formulate at way to easily tune the control variable mapping to a specific non-zero rate of failures, and this idea could not be studied. However the idea would need to be analyzed to determine if the added benefit of more aggressive assignment would outweigh the cost of essentially doubling the amount assignments needed to be made. In such a case the assignment would need to be at twice the performance to justify assignment at twice the rate.

Of note is that in order to implement this idea the performance option must be one or three, otherwise the added redundant cases will not contribute to the added stability, and the extra performance will come at the cost of added failures.

7.1.2 Error Analysis

This case will extend the standard problem to include certain errors in the system. Errors of omission or complete failure are represented in the system already, by removal of agents and modification of the communication network. The additional errors included will affect the cost matrix as it is communicated throughout the system. These errors are different as they can be represented by some small change in the true signal. The other existing errors are operating on discrete systems and can therefore only be represented by on or off. The new errors will be real in nature and will therefore shift the true data values. This problem has been proven to be highly sensitive to information agreement, and errors in this system will only stress that sensitivity. The true signal will be confounded by some error. The final signal is represented below in Equation 41. In this equation TS represents the true signal while Ω(X) represents the error or noise added. In this case the noise function is given by uniform distribution centered at zero and extending to plus or minus x. Cost values cannot be reduced below zero from error.

Equation 41 Error Signal

$$E = TS + \Omega(X)$$

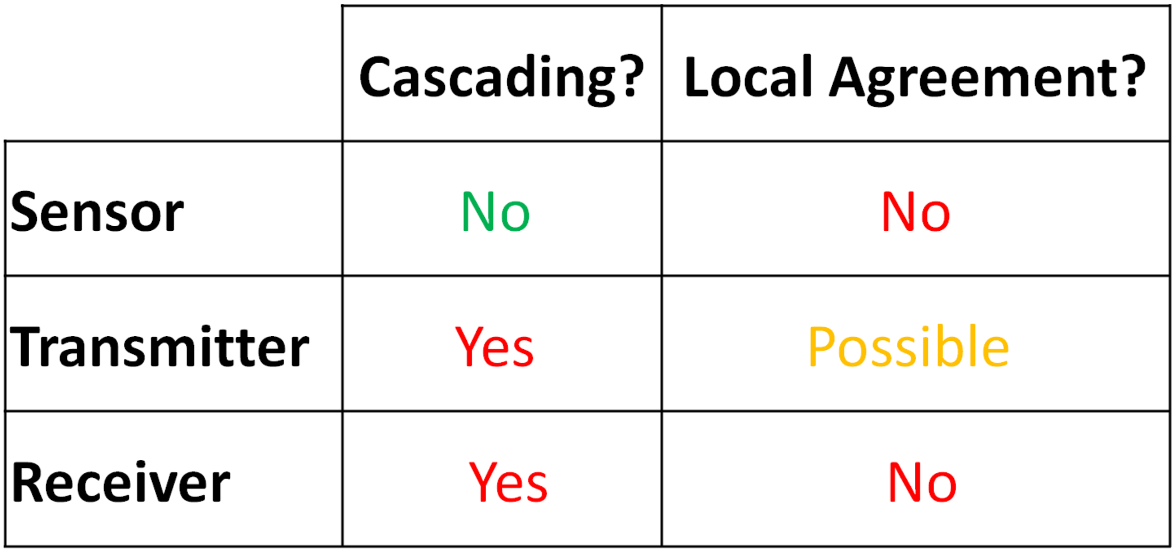
As these errors will operate on either parts of or the whole cost matrix, the above equation will be applied independently to each element affected. For each time step a new instantiation of the error function will be chosen such that the value of the signal will change at every time step, even if the true signal remains constant. This was chosen as it perhaps the most difficult possibility. If the error function is constant and only changes with each change to the true signal than the system would be operating on false information, but agreement would not be as affected as much. Three types of errors are considered, and are listed below:

- Sensor error: This error affects each agent's knowledge of how well in can make assignments with each of the compatible tasks.
- Transmitter error: This error acts on information that is transmitted from one agent to each of its neighbors. Even though the transmitted information contains errors, each of the neighboring agents will receive the same information.
- Receiver error: This error acts on the information that is received via communication from another agent. This is different from the previous type of error because two agents may be neighbors to the same third agent, but can receive different information even if the third agent has no transmitter error.

The vast majority of the problems from these types of errors arise from the fact that agreement is disrupted, rather than the fact that the system is acting on an error filled cost matrix. If the cost matrix had some errors, but each agent was in agreement on what the cost matrix was, than a proper assignment scheme could be made without failures, but it would be less optimal. However a less optimal solution is still superior to one that has failures or is the baseline assignment. Another way to think about these errors is to consider the change of meaning of the pre-consensus state. In the standard formulation, once the system reaches pre-consensus, than all agents will have agreement about the system information, but the lack of assuredness of this information keeps it from being a true consensus. Now, the pre-consensus agreement is that the major changes that have occurred in the system have been agreed upon, but the values of the cost matrix could vary greatly between different agents.

There are two major factors that these errors create which hurts the agreement and therefore hurts the overall assignment performance: Cascading effects and local agreement. The nature of the communication network means that some information will have to travel between multiple agents before it can be received by all agents in the system. If some error is added along each step, than that error will cascade, or increase as the number of agents it must go through increases. Think of this like a game of telephone. The longer the chain is, the more likely errors from each person will accumulate until the information is barely like what it was at the beginning. If the telephone chain was short, like two people then the outgoing message is more likely to be closer to the original when compared to a longer chain. Local agreement is also an issue with these errors. If and agent sends out information to its neighbors, will these neighbors have agreement as to what that information is? These two effects are considered for each type of error below in Table 93. Sensor error is the only type that does not cascade, as it is the only error type not directly involving communication. The transmitter error is slightly better behaved than the receiver error in that the former will give the same information to each of its neighbors, while in the case of the latter, the neighboring agents will not have agreement. This is not a major issue though, because it may be unlikely that neighbors of a specific agent will receive all of their information from that one agent rather than multiple agents. In the latter case, the effect of local agreement will be minimized because each of the agents supplying information will be subject to their own (different) error in transmission.

Table 93 Impacts of Different Error Types



	Cascading?	**Local Agreement?**
Sensor	No	No
Transmitter	Yes	Possible
Receiver	Yes	No

In order to study these errors, the standard test was modified to include each type of error independently. Without modification to the cooperative control scheme these errors provided dramatic reductions in performance as well as the number of failures in the system. Two figures have been reproduced with exaggerated results to show the abnormal behavior the errors caused (transmitter error is shown as it does not suffer from the cascading effect). This shows the effect of the error better than the three types confounded.

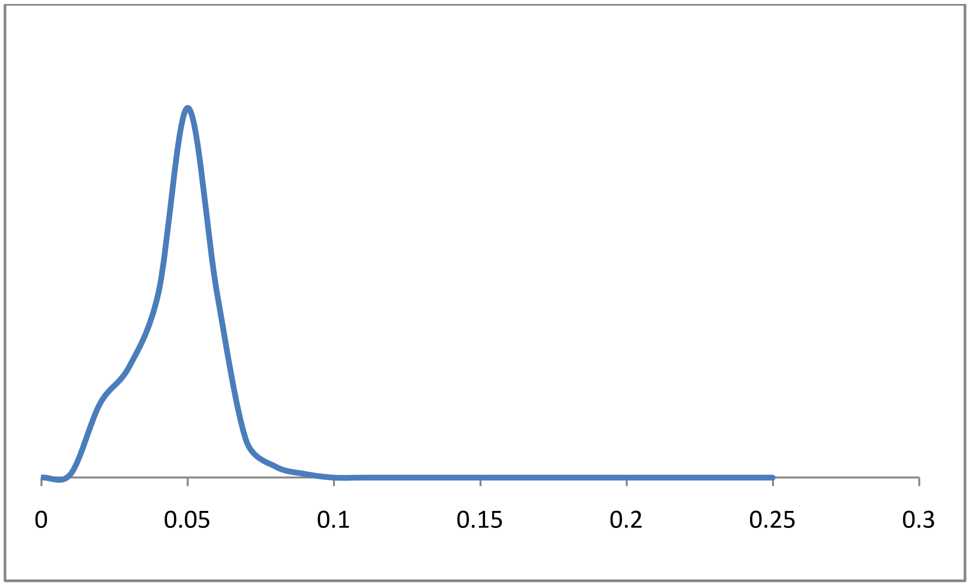


Figure 106 System Failure Rate Vs Strength of Error

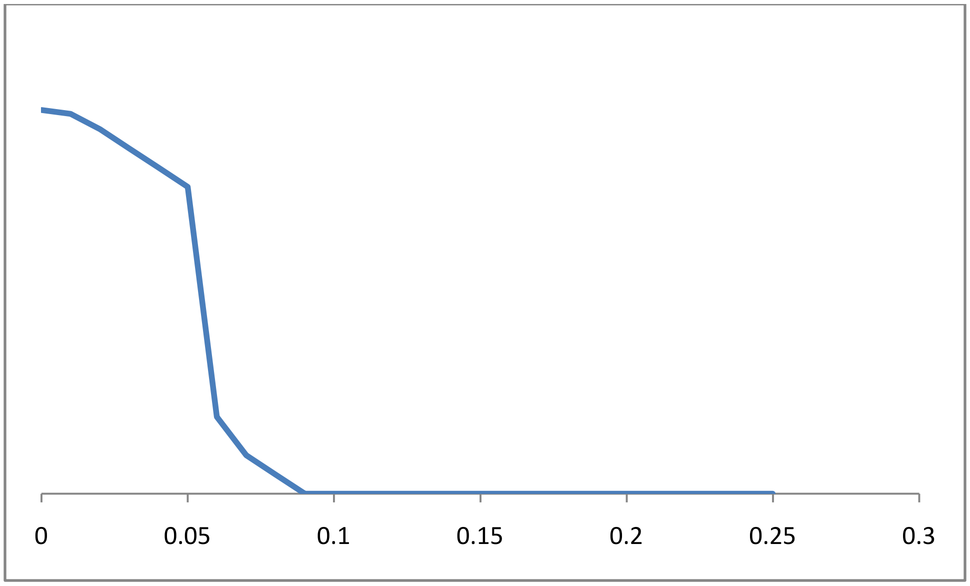


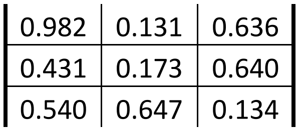
Figure 107 System Performance Vs Strength of Error

The most curious aspect of this is the fact that at a certain point the failures from error are essentially removed at the cost of performance being equal to the baseline case. This is due to the cost threshold variable introduced to help mitigate failures from cost changes. With error introduced, the changing cost is also affected by this variable. Each agent has no way of knowing whether or not the changes it is sensing or receiving via communication are changing because the true signal is changed or errors are causing this change. Once the error function reaches a value of .05 it becomes more and more likely that errors will trigger the cost threshold and cause the system to revert to the baseline control scheme. At larger values of the error function this is almost assured.

Other than this unusual behavior, this shows that the general agreement methods that work for the standard case must be modified to account for error. This is mainly to help agreement which will increase performance and decrease failure. Three options were considered to help reach agreement, but each comes at some cost. Those options are: reducing cost to bins; waiting for convergence of information before leaving the baseline control scheme and increasing the amount of redundancy in the system from the control mapping.

The first option involves a modification to the cost matrix to sacrifice some level of granularity and overall optimality to help agreement. This was the placing of cost values into bins which were more discrete than the continuous cost as it exists in the standard case. By placing the cost into bins, this should neutralize the errors if they are small enough. An example of putting cost into bins is given below. Take an initial cost matrix given by Table 94.

Table 94 Example Cost Matrix for Cost Bin Idea



0.982	0.131	0.636
0.431	0.173	0.640
0.540	0.647	0.134

Each cells non-zero value will be placed into three bins. Any value less than .33 will be represented by .25. Any value between .33 and .67 will be represented by .5.

Any value above .67 will be represented by .75. Once this transformation has taken place, Table 94 becomes Table 95 given below.

Table 95 After Bin Reduction

0.750	0.250	0.500
0.500	0.250	0.500
0.500	0.500	0.250

Now take the cost matrix of and add an error of size .1. The resulting cost matrix is given below in Table 96.

Table 96 Example Cost Matrix With Error

0.962	0.057	0.519
0.375	0.009	0.564
0.499	0.531	0.046

Now if the same bin reduction scheme is applied to the cost matrix with errors, the cost matrix becomes that given in Table 97 below. Notice that this new matrix happens to be the same as the reduced matrix without errors!

Table 97 Error Cost Matrix after Bin Reduction

0.750	0.250	0.500

0.500	0.250	0.500
0.500	0.500	0.250

While this idea sounds good in practice it did not work in this case (failures were not reduced), and the reason is somewhat simple. While the bin assignment makes errors less likely to affect the cost matrix, errors that do have a greater effect than they would otherwise. For example, a cost value close to the range of bin assignment could go either way just as likely, and instead of dealing with two agents having a different view of the cost matrix value which differs by .1, now it may vary by .5. This did not prevent failures; therefore this option was not implemented in the final control scheme. However, the next two options for increased agreement were implemented.

The next change implemented to tackle the lack of agreement focuses on the statistical nature of the error. If there is enough time between changes of the true function, the error will simply be comparing random samplings of the error function. The nature of each error function is that the expected error is zero, which is important to this formulation. By the Lindeberg-Levy Central Limit Theorem, the average distribution created by taking the mean of the samples is a well behaved normal distribution. This is demonstrated below by Equation 42. This requires a sufficiently large amount of samples of course.

Equation 42 Lindeberg-Levy Central Limit Theorem

$$\sqrt{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu\right) \rightarrow N(0, \sigma^2)$$

For the sake of this problem, this leads to an interesting result. If enough samples are taken (between changes of the true signal) than by averaging these values, the average should eventually be relatively constant and approach zero. The sample average is given below in Equation 43.

Equation 43 Sample Average

$$S_1 = X_1$$

$$S_n = \frac{(n-1)(S_{n-1}) + X_n}{n}$$

This means that if two consecutive values of the sample mean are compared and the difference is small enough, than the sample mean can be assumed to be the true signal with reasonable accuracy. This convergence criterion is given below in Equation 44. The value of .01 was chosen as it gives a good system behavior in terms of failures and minimum effective task performance.

Equation 44 Convergence Requirement of Sample Mean

$$S_{n-1} - \frac{(n-1)(S_{n-1}) + X_n}{n} < .01$$

What does this mean for the system? The amount of error in the system can be effectively reduced to nothing at the cost of spending a greater amount of time in the baseline assignment scheme. As long as the criterion in Equation 44 is not met, then the amount of agreement in the system is not enough to perform a more aggressive assignment. This can greatly increase the amount of time required without changes before pre-consensus can be reached, but it also reduces failures.

Increasing redundancy to an acceptable level from the standard case only involved taking the second most aggressive bin and expanding that to include the most aggressive bin. That is to say the modification is to assign bin one to the same control variables as bin two. This in addition to the next change seemed to be enough. It was the goal to change the control mapping as little as possible and take care of the error by other means. This preserves the quality of the control mapping as much as possible which should not reduce performance by too much from the mapping itself.

With these changes implemented, the errors are tested. Due to the nature of convergence, the standard testing scheme rarely was able to enter pre-consensus due to the rapidity of the changes occurring in the system. Instead, the most lax testing (given in Appendix B) was used such that convergence can occur. The results of each error independently are given below in Figure 108 and Figure 109.

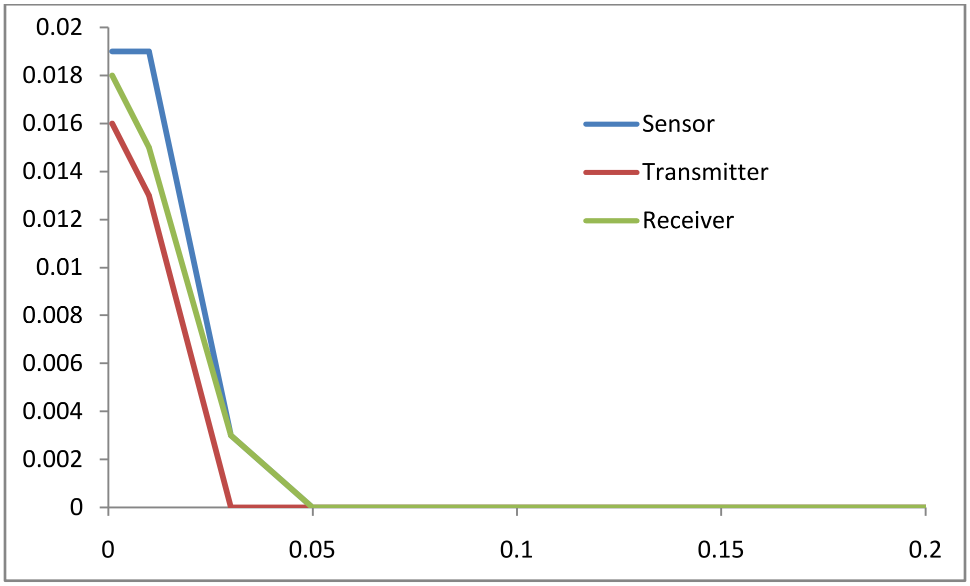


Figure 108 Average Effective Task Performance vs Size of Error

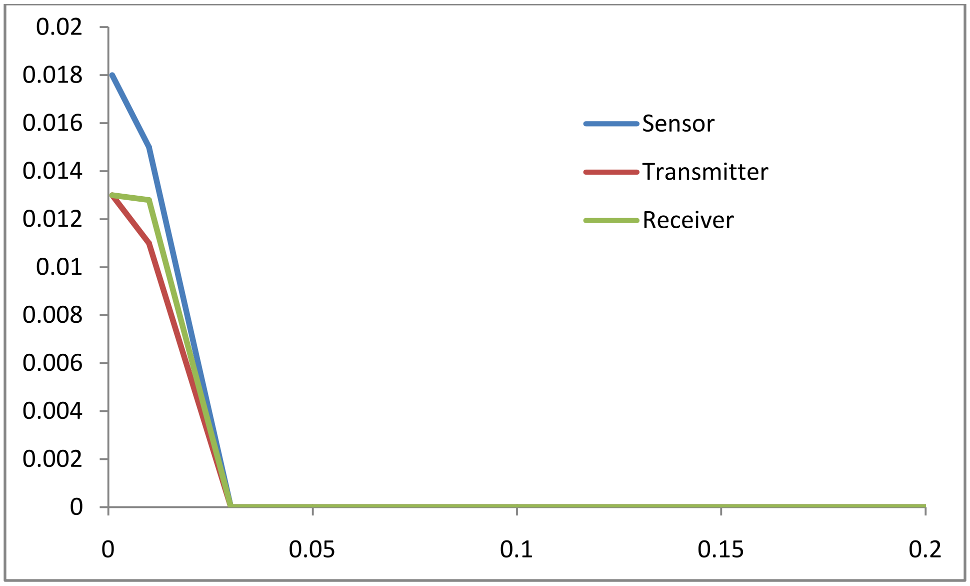


Figure 109 Minimum Effective Task Performance vs. Size of Error

As can be seen from these figures, for small enough error the system performance is about the same as it was without error. Low but non-zero errors serve to increase the time it takes to pre-consensus due to the added process of convergence of information. As error increases to about .05 the system does not have enough time to reach convergence except in rare cases. In addition to this, the cost threshold is low enough compared to the cost that the error can trigger a system reset as an error sample can be larger than the threshold. For the minimum effective task performance, the system effectively reaches baseline performance at a value less than .05 as this is a view of the worst case scenario. In both cases, the sensor error is less problematic than the transmitter and receiver error due to the nature of cascading error.

The main issue which reduces system performance is the amount of time it takes to reach pre-consensus. Once pre-consensus is reached, the system performance should be the same as it is without error. However it is not always that simple; the extra time needed to reach pre-consensus might not be small enough, as changes which are more frequent than this time will prevent pre-consensus. If pre-consensus takes five time steps to achieve without error, and changes occur every ten time steps, performance can increase. If errors cause pre-consensus takes fifteen time steps to be achieved with some amount of error, and changes occur every ten, than pre-consensus will not occur. To analyze this effect, the test bed was modified such that no system changes occur, and pre-consensus will occur. It may not be possible for pre-consensus to occur if the errors are so large that the cost reset threshold is met. The results of this study are shown below in Figure 110. In this figure, the time to pre-consensus is normalized by the time required with zero error.

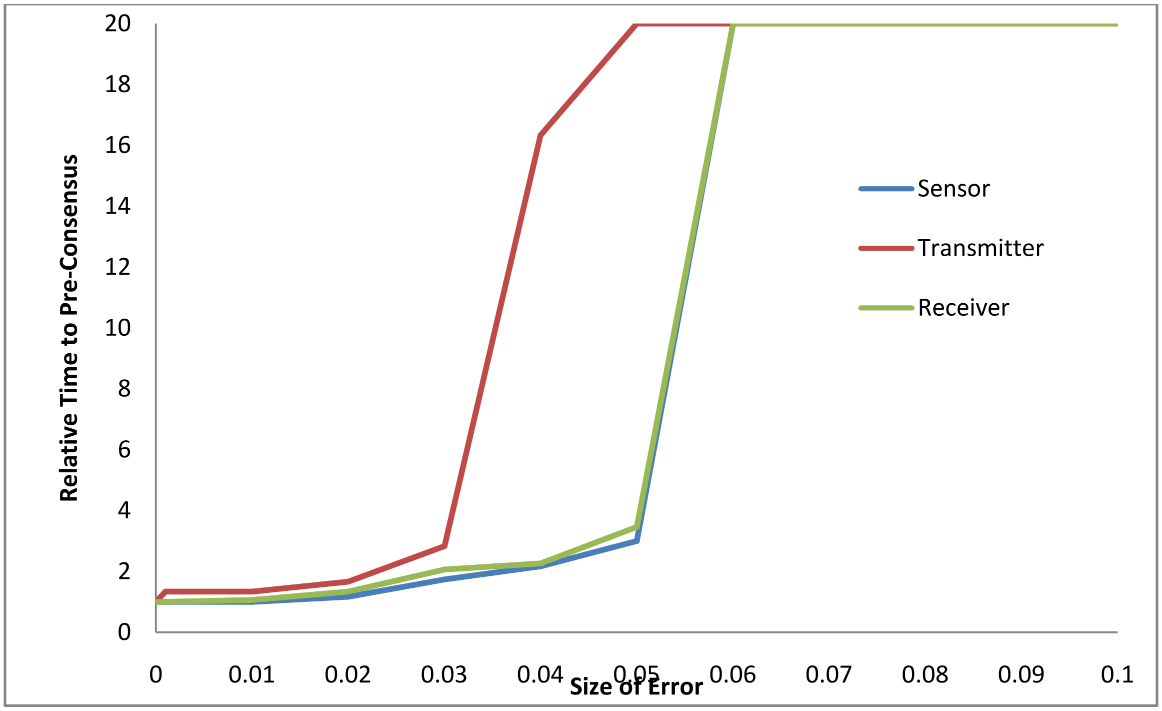


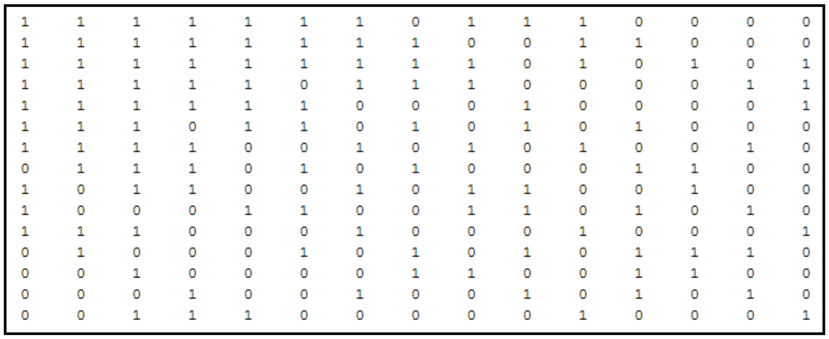
Figure 110 Relative Time to Pre-Consensus Based on Error

There are a few interesting points to make from this figure. Initially, the low levels of error, up to about .01, have about the same time to pre-consensus as cases without error. At .02 error, the increase in time to pre-consensus is about fifty percent. From this point, all times to pre-consensus begin to dramatically increase, the fastest increase being found by the transmitter error. The transmitter error seems to have the most impact in terms of cascading effects as compared to the other two error types. Finally, at some level the errors are too large and trigger the cost cutoff which leads to behavior reset. In these cases, the errors are too large and pre-consensus will never occur regardless of the frequency and impact of changes. In Figure 110, this was represented by a relative time value of twenty, which is the highest y-value in the figure.

The final message of this testing is that errors in the system can be accounted for at the cost of added time required in the baseline case before pre-consensus can be reached.

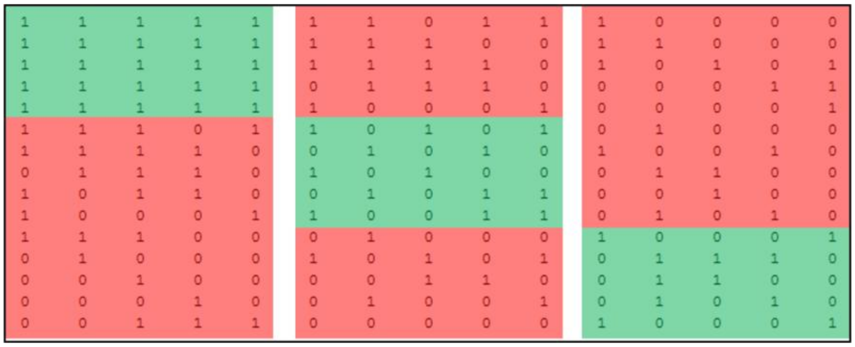
7.1.3 Smaller Team Analysis

This case will look at relaxing the maximum amount of agents and tasks in the system. The values were bounded due to reasons of computational resources, but it is very possible that in a real life system these values will exceed these bounds. This is problematic for one simple reason; the larger the system the larger the diameter. One measure of a well behaved system is that the diameter is much smaller than the size of the network. However the diameter should be as small as possible to help cover future problems. A modified test was created from the standard test, but changing the initial amount of agents and tasks to one hundred rather than their old values. After this testing, the system never entered pre-consensus, the diameters were too large to do so. What can be done to solve this problem? The diameter is what it is. The answer came from an unlikely place, and was partially mentioned before. When the system is disjointed or disconnected, it is essentially a number of individual smaller systems. These systems cannot communicate with each other, but the benefit is that the smaller networks result in smaller diameters, which will allow the system to more easily enter pre-consensus. It is possible to effectively separate the network while still allowing full communication by modifying the adjacency matrix before calculating the local network. This will still allow any and all communication which is possible to remain, while allowing the system to act as if it was disconnected. This can be seen below in Figure 111 and Figure 112.



1	1	1	1	1	1	1	0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	0	0	1	1	0	0	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
1	1	1	1	1	1	0	0	0	1	0	0	0	0	1
1	1	1	0	1	1	0	1	0	1	0	1	0	0	0
1	1	1	1	0	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	1	0	1	0	0	0	1	1	0	0
1	0	1	1	0	0	1	0	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0	1	1	0	1	0	1	0
1	1	1	0	0	0	1	0	0	0	1	0	0	0	1
0	1	0	0	0	1	0	1	0	1	0	1	1	1	0
0	0	1	0	0	0	0	1	1	0	0	1	1	0	0
0	0	0	1	0	0	1	0	0	1	0	1	0	1	0
0	0	1	1	1	0	0	0	0	0	1	0	0	0	1

Figure 111 Original Adjacency Matrix



1	1	1	1	1	1	1	0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	0	0	1	1	0	0	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
1	1	1	1	1	1	0	0	0	1	0	0	0	0	1
1	1	1	0	1	1	0	1	0	1	0	1	0	0	0
1	1	1	1	0	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	1	0	1	0	0	0	1	1	0	0
1	0	1	1	0	0	1	0	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0	1	1	0	1	0	1	0
1	1	1	0	0	0	1	0	0	0	1	0	0	0	1
0	1	0	0	0	1	0	1	0	1	0	1	1	1	0
0	0	1	0	0	0	0	1	1	0	0	1	1	0	0
0	0	0	1	0	0	1	0	0	1	0	1	0	1	0
0	0	1	1	1	0	0	0	0	0	1	0	0	0	1

Figure 112 Modified Adjacency Matrix

In the second figure, the system is reduced to three smaller networks, each of which will have a smaller diameter than the overall system. The problem now becomes what size should these groups be, he should these groups be formed, and what tasks should be grouped? The tasks have to be grouped as well otherwise the desired redundancy will be effectively much larger than it is desired to be. How to form these groups is a big problem. Before those issues will be discussed, a simpler test will be conducted. This test will eliminate the changes in number of agents and tasks, and only consider the changes in network and cost. This in addition to the increased size of the system is the only differences between this test and the standard test. For this problem, the system is reduced to ten teams of ten agents and ten tasks randomly chosen. The results of this test are given below in Table 98. It is important to note outside of the standard performance metrics, that the system went from performance better than the baseline case zero percent of the time using the traditional method vs. close to one hundred percent using the smaller teams idea. Changes are still occurring, but because the smaller teams may have different diameters, their behavior resets are staggered. This essentially means that not all of the small teams are affected by every change to the system, and most of them are on different coordination intervals in terms of reaching pre-consensus. Both of these facts help ensure that system performance is greater than the baseline case.

Table 98 Comparison of Results

	Traditional Method	Smaller Teams Implemented
Average Total	0	0.16
Average Minimum	0	0.13
Average Total	1	3.914
Average Minimum	1	2.105

The results are quite good, as pre-consensus can be reached and failures do not occur. The baseline normalization shows considerable improvement over the baseline case. Base on the average minimum performance increase, agents having half the ability would do slightly better than normal agents without cooperation. The reason for this is two-fold; achieving pre-consensus at all will increase performance, but the large size of the system further reduces the baseline case performance by requiring the agents split their assignment over as many as one hundred tasks (vs. a maximum of forty tasks in the standard case). These results give great promise for the idea of using smaller team sizes in systems of large size.

However the addition and removal of tasks and agents causes many more problems in the system. Even without those a major issue is the fact that there may be incompatibilities between the corresponding agent and task team. For a system of one hundred tasks and agents, some tasks might have a small number of compatible agents. Using random team assignment, it is more likely that the task will have no compatible agents in its team and therefore the system will fail. A number of issues arises as a result of the formation of smaller teams, and is given below:

- How to assign agents to teams?
    - Based on network proximity?
    - Based on shared task compatibility?
    - Based on differing task compatibility?
    - Some combination of the above?
- How large should the teams be?
    - A set number
    - A percentage of the overall system
- What to do with new agents
    - Join the first team they communicate with?
    - Wait till full knowledge of the system is known then make a decision?
- How to switch teams
    - As the system changes should the teams change?
    - Should this be based on the changes in the network, agents or tasks? Possibly all three?
- How to assign tasks to teams
    - Should tasks be assigned to multiple agent teams or single teams?
    - As tasks change incompatibility how should they change teams?

These problems are quite large and outside the bounds of this research. In addition, this creates two effective timescales of the system. Each smaller team will go into pre-consensus based on its own diameter. The overall system changes (like changing teams) must be coordinated in the system and therefore operate on the time scale of the overall network diameter. During this time the system is preparing to change due to other changes which might not even be important by the time the system is ready. This idea has some promise, but these issues are left for future research. Even if the system is large, the small team ability can be ignored and the system will not fail, it will just be less likely to achieve performance greater than the baseline case.

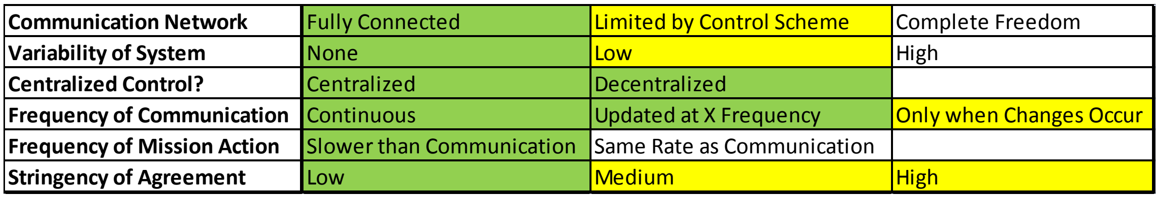
While this additional analysis does not fully address all of the problems of the initial cooperative control method, it gives avenues for how some of these problems can be alleviated. In addition, the periodic assignment case explores a situation in which agreement is made easier, and additional behaviors may be put in place to better conduct assignment for tasks which do not need to be covered for every time step. In the case of additional error being introduced, the system can handle this problem without an addition of failures, but at the cost of increased time to reach pre-consensus. Large systems are more likely to take longer to reach pre-consensus, and a case was presented which shows that in some cases splitting the overall system into smaller teams of agents may increase performance by making agreement easier to obtain. This case needs more study, as the problem introduced an interesting but difficult new class of problems, hierarchical systems of interacting cooperative control problems.

CHAPTER 8: CONCLUSIONS

The overall goal of this research was to both analyze and create an implementation for cooperative control to be applied in situations of high variability. This was difficult because it stretched many of the previous practices of cooperative control, specifically in regards to agreement. For a cooperative control scheme to be successful, it must have two things: a control scheme that can ensure proper behavior of the given agents in and the system and an effective way to ensure agreement amongst the agents to ensure that they are truly cooperating. As expected, having a high level of variability in the problem only made these goals more difficult to ensure. The communication aspect of the problem was focused on, due to its more widespread application to the overall field. The specific control problem in general has little applicability to other control problems.

The main options or considerations which are made in terms of communication and agreement issues are given below in (reproduced from chapter 1). The highlighted blue cells represent the options which are possible via existing techniques.

Table 5 Options Captured by Traditional CC Techniques



Communication Network	Fully Connected	Limited by Control Scheme	Complete Freedom
Variability of System	None	Low	High
Centralized Control?	Centralized	Decentralized	
Frequency of Communication	Continuous	Updated at X Frequency	Only when Changes Occur
Frequency of Mission Action	Slower than Communication	Same Rate as Communication	
Stringency of Agreement	Low	Medium	High

Immediately it can be seen that some options are just not handled via traditional techniques. High variability of the system is not handled by most existing techniques. In addition, while it may be possible in some cases, there is an inherent assumption in existing cooperative control problems. Centralized or decentralized control is a secondary consideration in many cases. Centralized control is generally easier to implement, but is more sensitive to changes in the system than decentralized control. This is because centralized control involves a hierarchical structure and losses can more easily disrupt the system.

The other factors are somewhat related to each other. In some respects, one or possible two more difficult features can be selected, at the cost of limiting the remaining factors. For example, if there is a high stringency of agreement (meaning system information needs strict rather than loose agreement), than the system needs to propagate information as quickly as possible throughout the system. This usually implies a fully connected network which cannot have variability, however other aspects of the problem may be allowed to change. Communication frequency can either be continuous (as often as possible) or when changes occur to ensure that changes are known about immediately.

A different example to consider would be one where the network is not fully connected. In order to ensure information agreement, consensus techniques are used. These techniques take some amount of time to reach a suitable level of agreement which may only asymptotically approach agreement rather than have absolute agreement. This directly implies a need for a lower stringency of agreement, as well as having mission action occur less often than communication. This will allow system agreement to occur before it needs to be acted upon. Static networks can allow for a reasonable level of flexibility in information agreement because its network properties are known and will not change. If the network is allowed some variability, than it will take longer to reach an agreement further limiting stringency of information. For consensus to work, the network must have some limitations on it, preventing full variability. Such requirements include needed to keep the network connected, and may also include connectivity requirements to minimize convergence time. It is desired in these cases that communication be continuous, however periodic can still be allowed, if the increased time for convergence is acceptable.

In contrast to these issues are the options which must be taken in order to solve a problem of high system variability. To reflect this, Table 6 is reproduced below (from Chapter 1).

Table 6 Options Needed to be Captured by Proposed Research

Communication Network	Fully Connected	Limited by Control Scheme	Complete Freedom
Variability of System	None	Low	High
Centralized Control?	Centralized	Decentralized	
Frequency of Communication	Continuous	Updated at X Frequency	Only when Changes Occur
Frequency of Mission Action	Slower than Communication	Same Rate as Communication	
Stringency of Agreement	Low	Medium	High

A few options are chosen in conjunction in this case which are not possible in the standard case. The main issue is variability, and it is allowed in many ways: agent addition or removal, task addition or removal, network changes and cost changes which represent a change of ability to complete desired tasks. These changes are allowed to occur as rapidly as communication occurs, and impact the entire system in the worst case. Any of these changes can cause failures via uncovered or unassigned tasks, which indicate the problem requires a high stringency of agreement. A high level of stringency is not possible with communication network with full variability. Even using consensus will not work, because during the time it takes to achieve an acceptable level of agreement, mission action is also occurring and this disagreement will cause issues. Other changes in the problem only exacerbate this issue. Finally, the freedom of the network means that no matter what, a true consensus cannot occur. Information from neighbors will be at least somewhat old when the agent needs to act using it. While this information can help performance if used properly, if it is not, performance will be worse than if each agent acts independently from the rest. Agent losses dictate the control scheme be decentralized, as any losses can cause failure of a centralized scheme. For the system to adapt properly in spite of these issues, continuous communication is chosen. Less frequent but scheduled communication is possible, but will increase the chance of system failures. Communication cannot be allowed to occur only when changes occur, because communication is not ensured to happen when an agent is removed from the system. Agent losses may occur rapidly without allowing communication to occur, and in such a case, the system would believe the lost agent was still there as it would never receive information to the contrary.

In order to tackle these issues, a new more relaxed idea of consensus was created. For this research, a pre-consensus is defined as the state when information agreement occurs, but the age of information implies that any future changes will be unknown for some amount of time. During the time it takes for that information to be propagated, redundancy must be used to prevent failures from occurring. When the system is not in pre-consensus, special behavior must be used to reduce failures. The high stringency of agreement for this problem means that any sort of aggressive control strategies when some disagreement exists will greatly increase the change of failure. Therefore when the system is not in pre-consensus, the system should revert to maximum redundancy which will help prevent failures. The transitioning between these states while minimizing failures is essentially the crux of the control technique.

The secondary need to provide good control performance in this variable system is the need to base the aggressiveness of the control strategy on the communication network. The communication network is a measure of how rapidly the system can respond to changes; the more compact the network, the more aggressive it can be because it can respond to changes faster. The compactness of the network is measured via three system variables: diameter; connectivity and Cheeger constant. Using these metrics, a system mapping was created and tuned to effectively obtain a desired redundancy for assignment from these network measures. This mapping was created with the main goal of reducing system failures via unassigned tasks. This makes the control scheme an adaptive one, based directly on the communication network. If such an adaptive control technique was not implemented, then the performance of the system would greatly suffer. Some networks are so sparse, that a more aggressive assignment scheme should not be pursued because the amount of time it takes to react to changes mean that failures are likely. In order to prevent failures, the redundancy must be based on the worse possible case, which would dictate maximum redundancy at all times, which is given performance equal to the baseline case. This would essentially wrap a fancy control technique around a difficult problem which gives nothing. Such a case gives performance obtained (more easily) by ignoring cooperation entirely.

To test the performance of the described control scheme, a test bed was created. The goal of this test bed is to allow the various changes that could occur and analyze the performance to make sure that failures are not more likely to occur in some situations. Various network types were included in this case as well as change options: including impact and frequency of changes. For the standard case, changes were allowed to occur as often as every time step, while impacting all possible members. This implies that in one time step, the system may lose all but one agent, which is highly impactful. Despite the highly taxing series of tests, the system achieved the goal of performance at least as good as the baseline case, which met the desired goal of the research. Performance was low, because the high impact test case meant that oftentimes changes were occurring more rapidly than the system could react. Despite this, a performance increase of about five percent was obtained compared to the baseline case. To more fully test the system, two other cases were created to make the impact of such changes more gradual, but still allowing them to occur. Dramatic changes were still allowed to occur, but over a certain length of time rather than immediately. In such cases performance dramatically increased. In the most lax case, performance of up to fifty percent compared to the baseline case.

There are a few downsides to the system which must be discussed. The first is that system failures cannot be ensured to be zero. Removal of agents cannot be fully accounted for, although their occurrence can be made arbitrarily small. The good news is that the other changes possible in the system can be neutralized via the pre-consensus implementation. The other major downside of the system is that its performance is reduced when changes are not as likely to occur. The main goal of the system is to use as much redundancy as is needed to prevent failures. This level of redundancy is likely to be too much in most cases, but just what is needed when certain low likelihood events do occur. Compared to the more static cases of traditional research, this loss of performance may be unpalatable. However in such cases, a compromise may be made. During pre-consensus, strategies may be as aggressive as desired. Changes may not be expected to occur, but when they do, the system may transition to the highest redundancy state. This would neutralize all changes except those of agent loss. For agent loss, the transition of behavior would only decrease system failures, but may not prevent them entirely. If the idea of pre-consensus is not implemented, any change may cause system failures which do nothing to increase performance. In this way, part of this research can be used in some existing techniques under the chance that some changes may occur. In this formulation, the added change in behavior when those changes occur would only increase performance and reduces failures. Perhaps the biggest downside to this method is the inability to ensure that the ideas discussed will have widespread applicability to other control problems. In this problem, the system is relatively agile in that it can change the control (via redundancy) in order to respond to changes. In other problem types, this may not be the case and needs to be studied.

Finally the system was tested at various other conditions to look at other problem types which may cause difficulties for the presented cooperative control scheme. The assignment of periodic tasks led to cases of increased performance, and gave an avenue for failure analysis and redundant systems to make their way into the assignment. Analyzing communication and sensing errors gave a solution in which the lag time required to enter pre-consensus was increased, but failures kept essentially zero. The scaling of the system proved to be an issue, as larger systems would likely have larger diameters and take more time to respond to failures than smaller systems. Initial analysis proved that this idea has merit, but additional problems must be tackled to analyze cases of hierarchical cooperative control problems.

8.1 Contributions

The contributions from this research come as a result from extending cooperative control to accommodate cases of high system variability. This variability is most problematic in terms of system agreement; therefore the contributions provided tackle that issue as well as increase system flexibility.

Perhaps the greatest contribution was one not originally part of the proposed research. Pre-consensus allows an idea of agreement to exist in a system when a true and known agreement will never be possible (due to information age). Using pre-consensus, the agents in the system can understand when it is possible to be aggressive and trust the old information from fellow agents, and when it is likely that operating on such information will yield failures. Doing so yields a simple technique to change the basic strategy from conservative to aggressive based on the detected changes from each agent.

The next contribution allows the system to dictate how aggressive the control scheme should be during pre-consensus. This is done by obtaining key network parameters based on the compactness of the communication, and mapping those values into the control metric, in this case redundancy. By using the network itself as an input for the control scheme, the transition between aggressive and conservative assignment was made easily and efficiently. This concept gave good system performance when possible, while preventing failures when it was not. Finally, the cooperative control scheme gave good performance in a variety of highly variable and impactful test cases, and excelled when the changes were less frequent and less impactful. This is important, as it indicates that these methods can be used in more gradual systems which may only change occasionally. This is because good performance is gained without the downside of an increase in system failures.

8.2 Future Work

While this research made some contributions to the field of cooperative control, many improvements can be made and additional work remains in this area. The goal of this research was to prove that a cooperative control scheme can be developed which is no worse than using the same system without cooperation and better when possible. This occurs despite the many changes possible in the system. This research has effectively demonstrated that this type of cooperation is possible, but not that that it is optimally implemented. It is believed that many improvements can be made, from the determination of pre-consensus, to the control mapping. The basic means and ideas of implementing cooperative control in such an environment implemented in this research may also be improved upon. This research may be simply a means to open the door to much better future research in the field of cooperative control in highly variable systems.

Some research areas exist for new avenues of study rather than general improvements of what was presented. Perhaps the largest area for this is in the use of smaller teams. If this idea is to scale for larger systems, something additional must be done to improve performance. It is believed that the best idea is to split the larger system into smaller teams which will each handle a subset of the total tasks. This is difficult because it adds another level of coordination beyond the control task of the smaller teams. This is essentially creating a system of cooperative control problems which lightly interact, and changing those via a larger cooperative control problem. This research tackled at best system of equal cooperative control problems but not a hierarchical system of them. The major issues to tackle in this area are:

- How to form teams of agents
    - How to assign groups of tasks to groups of agents
- Means to change teams and strategy overall
- Coordination timing issues between sub-level control and team changes.

The next major area for improvement is in dealing with the cost and determining which changes in the cost matrix are important or impactful and which aren't. Early analysis in this research led to somewhat brute force techniques, while it is believe that some method must exist to better identify problematic cells of the matrix to better analyze changes. This would allow for increased performance by staying in pre-consensus when changes are not enough to cause failures and can be handled via redundancy until the impact of the change is known.

The final and perhaps most important piece of future work is the application of lessons learned in this research to more specific problems. The goal of this research was to analyze many types of networks, system configurations and changes, but real problems will likely be much more focused. Taking into account the specifics of the problem will allow more focused understanding of the control and it will be better suited for that problem. Another issue which will likely arise are problems in which there is no easily defined baseline to fall back to when changes occur. This research can describe when to know when coordination is unlikely, but what can be done in those situations?

Appendix A Data Ranges for Testing

	High Impact (Nominal)	
	Min	Max
# Initial Agents	5	20
# Initial Enemies	5	20
Add/Remove set interval	1	6
Add/Remove Task Set Interval	1	6
Modify Links Set Interval	1	6
Change Cost Set Interval	1	6
Change Detected On	0	1
Network Type	1	4
Network Type2	0	0.5
Network Properties 2	0	0.5
Network Properties 3	0	0.5
Add/Remove flagadd	3	3
Add/Remove flagtime	1	1
Add/Remove whichtoremoveflag	1	3
Add/Remove minadd	0	0
Add/Remove maxadd	0	0.9
Add/Remove addchance	0	1
Add/Remove totalmin	2	5
Add/Remove totalmax	10	20
Add/Remove linkmin	0	5
Add/Remove linkmax	5	10
Add/Remove Task flagadd	1	1
Add/Remove Task flagtime	1	1
Add/Remove Task whichtoremoveflag	1	1
Add/Remove Task minadd	0	5
Add/Remove Task maxadd	5	10
Add/Remove Task chance	0	1
Add/Remove Task totalmin	5	10
Add/Remove Task totalmax	20	40
Add/Remove Task linkmin	0.25	0.5

Add/Remove Task linkmax	0.5	1
Add/Remove Task linkaddflag	2	2
Modify Links flagtime	1	1
Modify Links addflag	1	3
Modify Links addflagnumber	3	3
Modify Links addchance	0.25	1
Modify Links minadd	0	0.5
Modify Links maxadd	0.2	0.9
Modify Links removeflag	1	3
Modify Links removeflagnumber	3	3
Change Cost flagtime	1	1
Change Cost flagchange	1	2
Change Cost minchange	0	0
Change Cost maxchange	0.1	0.9
Change Cost flaghowtochange	1	1
Change Cost amount to change min	0	0
Change Cost amount to change max	0.1	0.9
Min cost	0.1	0.45
Max cost	0.55	0.9

Appendix B Additional Values for Testing (Lax Testing)

Highlighted Cells Demonstrate Differences from Appendix A

	Medium Impact		Low Impact	
	Min	Max	Min	Max
# Initial Agents	5	20	5	20
# Initial Enemies	5	20	5	20
Add/Remove set interval	3	6	5	6
Add/Remove Task Set Interval	3	6	5	6
Modify Links Set Interval	3	6	5	6
Change Cost Set Interval	3	6	5	6
Change Detected On	3	6	5	6
Network Type	1	4	1	4
Network Type2	0	0.5	0	0.5
Network Properties 2	0	0.5	0	0.5
Network Properties 3	0	0.5	0	0.5
Add/Remove flagadd	3	3	3	3
Add/Remove flagtime	1	1	1	1
Add/Remove whichtoremoveflag	1	3	1	3
Add/Remove minadd	0	0	0	0
Add/Remove maxadd	0	0.5	0	0.1
Add/Remove addchance	0	1	0	1
Add/Remove totalmin	2	5	2	5
Add/Remove totalmax	10	20	10	20
Add/Remove linkmin	0	5	0	5
Add/Remove linkmax	5	10	5	10
Add/Remove Task flagadd	1	1	1	1
Add/Remove Task flagtime	1	1	1	1
Add/Remove Task whichtoremoveflag	1	1	1	1
Add/Remove Task minadd	0	5	0	5
Add/Remove Task maxadd	5	10	5	10
Add/Remove Task chance	0	1	0	1
Add/Remove Task totalmin	5	10	5	10

Add/Remove Task totalmax	20	40	20	40
Add/Remove Task linkmin	0.25	0.5	0.25	0.5
Add/Remove Task linkmax	0.5	1	0.5	1
Add/Remove Task linkaddflag	2	2	2	2
Modify Links flagtime	1	1	1	1
Modify Links addflag	1	3	1	3
Modify Links addflagnumber	3	3	3	3
Modify Links addchance	0.25	1	0.25	1
Modify Links minadd	0	0.25	0	0.05
Modify Links maxadd	0.1	0.5	0	0.1
Modify Links removeflag	1	3	1	3
Modify Links removeflagnumber	3	3	3	3
Change Cost flagtime	1	1	1	1
Change Cost flagchange	1	2	1	2
Change Cost minchange	0	0	0	0
Change Cost maxchange	0.1	0.5	0	0.1
Change Cost flaghowtochange	1	1	1	1
Change Cost amount to change min	0	0	0	0
Change Cost amount to change max	0.1	0.5	0.1	0.1
Min cost	0.1	0.45	0.1	0.45
Max cost	0.55	0.9	0.55	0.9

Appendix C Source Code

The simulation environment was created using matlab. In order to run the simulation, an excel spreadsheet of inputs must be created. Each of the variables found in the previous appendices are given values along each row of the spreadsheet. Each row represents a different case to be run. After all cases have been simulated, an excel spreadsheet is created as output for each case as well as a summary of all cases.

Appendix C.1 ADDREMOVEENEMYUNITS.M

```
function
[newenemymat,targettime,newcost,newenemycheck,enemytracker,totalenemies
,newenemytime,newenemytimeleft,newnocomenemytimeleft] =
addremoveenemyunits(mat,adjacency,enemymat,time,targettime,cost,mincost
,maxcost,alliedtracker,totalunits,newenemycheck,enemytracker,totalenemi
es,enemytime,enemytimeleft,nocomenemytimeleft,mintime,maxtime,onechance
)

flagadd = mat(1);        %dictates means of addition/removal of units
flagremove = mat(1);
setinterval   =   mat(2);         %dictates   amount   of   time   between
addition/removal and can be modified
flagtime = mat(3);
whichtoremoveflag=mat(4);
minadd = mat(5);         %minimum amount to be added
maxadd = mat(6);         %max amount to be added
addchance = mat(7);
removechance = mat(7);
totalmin = mat(8);       %total minimum number of units
totalmax = mat(9);       %total maximum number of units
linkmin = mat(10);       %Minimum number of links from allies to enemy
linkmax = mat(11);
linkaddflag = mat(12);

numunits=length(adjacency);
numenemies=size(enemymat);
numenemies=numenemies(2);
newenemymat=enemymat;
newcost=cost;
newenemytime=enemytime;
newenemytimeleft=enemytimeleft;
newnocomenemytimeleft=nocomenemytimeleft;
numallies=length(adjacency);

numadded=0;

%determine if number of units will be changed

if time>=targettime

    %Set New target time
    if flagtime == 1
        interval=setinterval;
    elseif flagtime == 2
        interval=randinterval(intlower,intupper);
    end

    targettime = time+interval;

    %remove units
    %determine number to remove

    if flagremove==1
        if rand<removechance
            numremoved=randinterval(minadd,maxadd);
        else
            numremoved=0;
        end
    elseif flagremove==2

    end

    removecheck=numenemies-numremoved;

    if removecheck<totalmin
        numremoved=numenemies-totalmin;
    end

    if numremoved>1

        if whichtoremoveflag==1
            removedunits=randperm(numenemies);
            removedunits=removedunits(1:numremoved);
        elseif whichtoremoveflag==2

        end

        removedunits=-sort(-removedunits);

        for i=1:length(removedunits)
            newenemymat(:,removedunits(i))=[];
            enemytracker(removedunits(i))=[];
            totalenemies(removedunits(i))=0;
            newcost(:,removedunits(i))=[];
            newenemytime(removedunits(i))=[];
            newenemytimeleft(removedunits(i))=[];
            newnocomenemytimeleft(:,removedunits(i))=[];
        end

    end

    %add units
    if flagadd==1
```

```
    if rand<addchance
        numadded=randinterval(minadd,maxadd);
    else
        numadded=0;
    end
elseif flagadd==2

end

addcheck=numenemies+numadded;

if addcheck>totalmax
    numadded=totalmax-numenemies;
end

numenemies=size(newenemymat);
numenemies=numenemies(2);

%%%%link new units
if numadded>0

    newenemymat(1:numunits,numenemies+1:numadded)=0;
    newcost(1:numunits,numenemies+1:numadded)=0;
    newnocomenemytimeleft(1:numunits,numenemies+1:numadded)=0;

    for i=1:numadded
        totalenemies(end+1)=1;
        enemytracker(end+1)=length(totalenemies);
        newenemytimeleft(end+1)=1;
        if rand<onechance
            newenemytime(end+1)=1;
        else
            dummyvalue=randinterval(mintime,maxtime);
            newenemytime(end+1)=dummyvalue;
        end
    end

    for i=1:numadded

        if linkmax>numunits
            linkmax=numunits;
        end

        %Means of adding links (amount or percentage)
        if linkaddflag==1
            linkminuse=linkmin;
            linkmaxuse=linkmax;
        elseif linkaddflag==2
            linkminuse=floor(numallies*linkmin);
            linkmaxuse=ceil(numallies*linkmax);
            end

            if linkmaxuse>numallies
                linkmaxuse=numallies;
            end
            if linkminuse<0
                linkminuse=0;
            end

            if linkminuse<3
                linkminuse=3;
            end

            linkdummy=randinterval(linkminuse,linkmaxuse);

            if linkdummy>0
                newpairs=randperm(numunits);
                if linkdummy>size(newpairs)
                    linkdummy=size(newpairs);
                end
                newpairs=newpairs(1:linkdummy);
                for j=1:linkdummy
                    newenemymat(newpairs(j),numenemies+i)=1;
                    newenemycheck(alliedtracker(newpairs(j)))=1;
                end
            else
                newenemymat(1,numenemies+i)=1;
            end
        end
    end
end

%Check for unlinked enemeis

[dummy1,dummy2]=size(newenemymat);

if dummy1>1
    numlinks=sum(newenemymat);
else
    numlinks=newenemymat;
end

if linkmin<1
    linkmin=1;
end

%Link enemies with minimum number of links
for i=1:numenemies
    dummycount=0;
    if numlinks(i)<linkmin
```

```
        for j=1:numunits
            if newenemymat(j,i)==0
                dummycount=dummycount+1;
                possiblepair(dummycount)=j;
            end
        end
        linksadded=linkmin-numlinks(i);
        if dummycount>0
            if dummycount<linksadded
                linksadded=dummycount;
            end
            whichtochoose=randperm(dummycount,linksadded);
            for j=1:linksadded
                newenemymat(possiblepair(whichtochoose(j)),i)=1;

newenemycheck(alliedtracker(possiblepair(whichtochoose(j))))=1;

newcost(possiblepair(whichtochoose(j)),i)=rand()*(maxcost-
mincost)+mincost;

newnocomenemytimeleft(possiblepair(whichtochoose(j)),i)=1;
            end
        end
    end
end

if numadded>=1
    for i=1:numallies
        for j=1:numadded
            if newenemymat(i,j+numenemies)==1
                newcost(i,j+numenemies)=(maxcost-
mincost)*rand()+mincost;
                newnocomenemytimeleft(i,j+numenemies)=1;
            end
        end
    end
end

end
```

Appendix C.2 ADDREMOVEUNITS.M

```
function
[newadjacency,newenemymat,targettime,newcost,newalliedtracker,newtotalu
nits,lastchange,newneighborknowledge,newenemycheck,newnocomenemytimelef
t] =
addremoveunits(mat,adjacency,enemymat,time,targettime,cost,mincost,maxc
ost,alliedtracker,totalunits,newenemycheck,lastchange,neighborknowledge
,enemytime,nocomenemytimeleft)

flagadd = mat(1);            %dictates means of addition of units
flagremove =mat(2);          %dictates means of removal of units
setinterval  =  mat(3);              %dictates  amount  of  time  between
addition/removal and can be modified
flagtime = mat(4);
whichtoremoveflag=mat(5);
minadd = mat(6);             %minimum amount to be added
maxadd = mat(7);             %max amount to be added
addchance = mat(8);          %chance for c
removechance = mat(9);
totalmin = mat(10);          %total minimum number of units
totalmax = mat(11);          %total maximum number of units
linkmin = mat(12);           %minimum number of links for new units
linkmax = mat(13);           %maximum number of links for new units
nominal = mat(14);           %Nominal % to

minunits=2;
maxunits=40;

addedlinksflag=1;

numunits=length(adjacency);
numenemies=size(enemymat);
numenemies=numenemies(2);
newadjacency=adjacency;
newneighborknowledge=neighborknowledge;
newenemymat=enemymat;
newcost=cost;
newalliedtracker=alliedtracker;
newtotalunits=totalunits;
newnocomenemytimeleft=nocomenemytimeleft;

totalunitcount=length(totalunits);

%determine if number of units will be changed
```

```
if time>=targettime
    %Set New target time
    if flagtime == 1
        interval=setinterval;
    elseif flagtime == 2
        interval=randinterval(intlower,intupper);
    end

    targettime = time+interval;

    %remove units
    %determine number to remove
    if rand<removechance
        if flagremove==1
            numremoved=randinterval(minadd,maxadd);
        elseif flagremove==2
            minremoved=floor(minadd*numunits);
            maxremoved=ceil(maxadd*numunits);
            numremoved=randinterval(minremoved,maxremoved);
        elseif flagremove==3
            minremoved=floor(minadd*nominal);
            maxremoved=ceil(maxadd*nominal);
            numremoved=randinterval(minremoved,maxremoved);
        end
    else
        numremoved=0;
    end

    removecheck=numunits-numremoved;

    if removecheck<totalmin
        numremoved=numunits-totalmin;
    end

    if numremoved>0
        if whichtoremoveflag==1
            removedunits=randperm(numunits);
            removedunits=removedunits(1:numremoved);
        elseif whichtoremoveflag==2
            removedunits=preferentialremoval(adjacency,numremoved,1);
        elseif whichtoremoveflag==3
            removedunits=preferentialremoval(adjacency,numremoved,2);
        end

        removedunits=-sort(-removedunits);

        for i=1:length(removedunits)
            newadjacency(removedunits(i),:)=[];
            newadjacency(:,removedunits(i))=[];
            newneighborknowledge(alliedtracker(removedunits(i)),:)=0;
            newcost(removedunits(i),:)=[];
            newenemycheck(removedunits(i))=0;
            newtotalunits(alliedtracker(removedunits(i)))=0;
            newenemymat(removedunits(i),:)=[];
            lastchange(removedunits(i))=[];
            dummycount=length(newalliedtracker);
            newnocomenemytimeleft(removedunits(i),:)=[];

            newalliedtracker(removedunits(i))=[];
        end
    end
    %add units
    if rand<addchance
        if flagadd==1
            numadded=randinterval(minadd,maxadd);
        elseif flagadd==2
            minadded=floor(minadd*totalmax);
            maxadded=ceil(maxadd*totalmax);
            numadded=randinterval(minadded,maxadded);
        elseif flagadd==3
            minadded=floor(minadd*nominal);
            maxadded=ceil(maxadd*nominal);
            numadded=randinterval(minadded,maxadded);
        end
    else
        numadded=0;
    end

    addcheck=numunits+numadded;

    if addcheck>totalmax
        numadded=totalmax-numunits;
    end

    numunits=length(newadjacency);

    newnumunits=numunits+numadded;

    pairlist=zeros(numadded,newnumunits);
    newenemymat(numunits+1:newnumunits,1:numenemies)=0;
    newcost(numunits+1:newnumunits,1:numenemies)=0;
    newnocomenemytimeleft(numunits+1:newnumunits,1:numenemies)=0;

    linkdummy=zeros(numadded,1);
    for i=1:numadded
        newtotalunits(end+1)=1;
        dumdum=length(newtotalunits);
        newalliedtracker(end+1)=dumdum;
        newenemycheck(end+1)=1;
        newneighborknowledge(dumdum,dumdum)=time+1;
        lastchange(end+1)=time;
        newnocomenemytimeleft(end+1,:)=0;
    end

    totalunitcount=totalunitcount+numadded;

    %%%%link new units
    if numadded>0
        for i=1:numadded
            if newnumunits<=linkmax
                linkmax=newnumunits-1;
            end
            linkdummy(i)=randinterval(linkmin,linkmax);
            if linkdummy(i)>0
                pairlist(i,:)=randperm(newnumunits);
            end
        end

        if length(linkdummy)>length(pairlist(1,:))
            pairlist(1,length(linkdummy):end)=[];
        end

        for i=1:numadded
            newadjacency(numunits+i,numunits+i)=1;

            if addedlinksflag==1
                if linkdummy(i)>0
                    [duma,dumb]=size(pairlist);
                    if linkdummy(i)>dumb
                        linkdummy(i)=dumb;
                    end
                    for j=1:linkdummy(i)
                        if numunits+i==pairlist(i,j)
                            pairlist(i,j:end-1)=pairlist(i,j+1:end);
                        end
                        newadjacency(numunits+i,pairlist(i,j))=1;
                        newadjacency(pairlist(i,j),numunits+i)=1;
                    end
                end
            elseif addedlinksflag==2

newadjacency=preferentiallinkaddition(adjacency,linkdummy(i),1);
            elseif addedlinksflag==2

newadjacency=preferentiallinkaddition(adjacency,linkdummy(i),2);
            end
        end

        %Link New Units to Enemies
        for i=numunits+1:newnumunits
            linkdummy=randinterval(1,numenemies);

            newpairs=randperm(numenemies);
            if linkdummy>0
                for j=1:linkdummy
                    newenemymat(i,newpairs(j))=1;
                    newnocomenemytimeleft(i,newpairs(j))=1;
                end
            end

        end


        %Update Cost
        for i=numunits+1:newnumunits
            for j=1:numenemies
                if newenemymat(i,j)==1
                    newcost(i,j)=(maxcost-mincost)*rand()+mincost;
                end
            end
        end


    end
end

end
```

Appendix C.3 ASSIGNMENTFROMX.M

```
function
[assignment,matching,assigned,unassigned]=assignmentfromx(x,numallies,n
umenemies,cost)

if numallies>numenemies
    numenemies=numallies;
    firstassignment=zeros(numallies);
else
    firstassignment=zeros(numallies,numenemies);
end
assignment=zeros(numallies,numenemies);
assignstore=assignment;
unassigned=[];
assigned=[];
matching=zeros(1,numallies);

for i=1:numallies
    for j=1:numenemies
        assignstore(i,j)=x(j+numenemies*(i-1));
        if x(j+numenemies*(i-1))>=.6
            if cost(i,j)<=-100
            else
                firstassignment(i,j)=1;
                matching(i)=j;
                break
            end

        end
    end
end

for i=1:numel(matching)
    if matching(i)>0
        j=matching(i);
        assignstore(i,:)=-100;
        assignstore(:,j)=-100;
    end
end

for j=1:numenemies
    if ismember(matching,j)==0
        dummy=max(assignstore(:,j));
        if dummy>0
            repeatcheck=-sort(-assignstore(:,j));
            if repeatcheck(1)>repeatcheck(2)
                for i=1:numallies
                    if assignstore(i,j)==dummy;
                        if cost(i,j)<=-100
                        else
                            firstassignment(i,j)=1;
                            matching(i)=j;
                            assignstore(i,:)=-100;
                            assignstore(:,j)=-100;
                        end
                    end
                end
            end
        end
    end
end

counter=1;
counter2=1;
if numenemies<=numallies
    for i=1:numallies
        if sum(firstassignment(i,:))==0
            unassigned(counter)=i;
            counter=counter+1;
        else
            assigned(counter2)=i;
            counter2=counter2+1;
        end
    end
elseif numenemies>numallies
    for i=1:numenemies
        if sum(firstassignment(:,i))==0
            unassigned(counter)=i;
            counter=counter+1;
        else
            assigned(counter2)=i;
            counter2=counter2+1;
        end
    end
end
assignment=firstassignment(:,1:numenemies);


if numel(unassigned)>numenemies
    uncoveredenemies(1)=0;
    uncovcounter=0;
    %Determine Which enemies are unassiged
    for i=1:numallies
        if sum(ismember(matching,i))==0
            uncovcounter=uncovcounter+1;
            uncoveredenemies(uncovcounter)=i;
        end
    end

    for i=1:numel(uncoveredenemies)
        [dum,loc]=max(assignstore(:,uncoveredenemies(i)));
        matching(loc)=uncoveredenemies(i);
        assignstore(loc,:)=0;

    end
end

end
```

Appendix C.4 ASSIGNMENTFROMXREMAINDER.M

```
function
[assignment,matching,assigned,unassigned]=assignmentfromxremainder(x,nu
mallies,numenemies,matching,count,cost)

if numallies>numenemies

    firstassignment=zeros(numallies);
    assignment=zeros(numallies,numenemies);
    assignstore=zeros(numallies,numenemies);
    unassigned=[];
    assigned=[];

    for i=1:numallies
        for j=1:numallies
            assignstore(i,j)=x(j+numallies*(i-1));
            if x(j+numallies*(i-1))>=.501
                if cost(i,j)<=0
                else
                    firstassignment(i,j)=1;
                    matching(i)=j;
                    break
                end
            end
        end
    end

    for i=1:numel(matching)
        if matching(i)>0
            j=matching(i);
            assignstore(i,:)=-500;
            assignstore(:,j)=-500;
        end
    end

    for j=1:numenemies
        if ismember(matching,j)==0
            dummy=max(assignstore(:,j));
            if dummy>0
                repeatcheck=-sort(-assignstore(:,j));
                if repeatcheck(1)>repeatcheck(2)
                    for i=1:numallies
                        if assignstore(i,j)==dummy;
                            if cost(i,j)<=0
                            else
```

```
                                firstassignment(i,j)=1;
                                matching(i)=j;
                                assignstore(i,:)=-500;
                                assignstore(j,:)=-500;
                            end
                        end
                    end
                end
            end
        end
    end

    counter=1;
    counter2=1;
    for i=1:numallies
        if matching(i)~=0
            assigned(counter)=i;
            counter=counter+1;
        else
            unassigned(counter2)=i;
            counter2=counter2+1;
        end
    end

    assignment=firstassignment(:,1:numenemies);

else

end
```

Appendix C.5 CHANGECOST.M

```
function [cost,targettime] = changecost(mat,cost,targettime,time)

flagtime = mat(1);
flagchange = mat(2);
setinterval = mat(3);
minchange = mat(4); %Either % or set amount
maxchange =  mat(5);
flaghowtochange = mat(6);
amounttochangemin = mat(7); %Percentage or amount
amounttochangemax = mat(8);
minvalue = mat(9);
maxvalue = mat(10);

[numallies,numenemies]=size(cost);

if time>=targettime
    if flagtime == 1
        interval=setinterval;
    elseif flagtime == 2
        interval=randinterval(intlower,intupper);
    end

    targettime = time+interval;

    nonzerocost=0;
    whichnonzero(1).located(1:2)=0;
    for i=1:numallies
        for j=1:numenemies
            if cost(i,j)>0
                nonzerocost=nonzerocost+1;
                whichnonzero(nonzerocost).locate(1)=i;
                whichnonzero(nonzerocost).locate(2)=j;
            end
        end
    end

    numnonzero=nonzerocost;

    %Determine Amount to change
    if flagchange==1
        minc=minchange;
        maxc=maxchange;
    elseif flagchange==2
        minc=floor(minchange*nonzerocost);
        maxc=ceil(maxchange*nonzerocost);
    end

    changeamount=rand*(maxc-minc)+minc;

    changeamount=round(changeamount*nonzerocost);

    if changeamount>nonzerocost
        changeamount=nonzerocost;
    end

    %Determine Which to Change
    dummy=randperm(numnonzero);
    tochange=dummy(1:changeamount);

    %Determine How Much to change
    if flaghowtochange==1 %Set amount
        for i=1:changeamount
            changeies=rand()*(amounttochangemax-
amounttochangemin)+amounttochangemin;
            dummy=rand();
            costi=whichnonzero(tochange(i)).locate(1);
            costj=whichnonzero(tochange(i)).locate(2);
            coststore=cost(costi,costj);
            if dummy<=.5
                coststore=coststore-changeies;
            else
                coststore=coststore+changeies;
            end
            if coststore>maxvalue
                coststore=maxvalue;
            elseif coststore<minvalue
                coststore=minvalue;
            end
            cost(costi,costj)=coststore;
        end
    elseif flaghowtochange==2
        for i=1:changeamount
            dummy=rand();
            costi=whichnonzero(tochange(i)).locate(1);
            costj=whichnonzero(tochange(i)).locate(2);
            coststore=cost(costi,costj);
            min=amounttochangemin*coststore;
            max=amounttochangemax*coststore;
            changeies=rand()*(max-min)+min;
            if dummy<=.5
                coststore=coststore-changeies;
            else
                coststore=coststore+changeies;
            end
            if coststore>maxvalue
```

```
                coststore=maxvalue;
            elseif coststore<minvalue
                coststore=minvalue;
            end
            cost(costi,costj)=coststore;
        end
    end
end
end
```

Appendix C.6 COMMUNICATIONSTAGE.M

```
function
[lastchange,newneighborknowledge,activeenemies]=communicationstage(adja
cency,lastchange,neighborknowledge,alliedtracker,time,cost)
oldneighborknowledge=neighborknowledge;
[numallies,numenemies]=size(cost);

for i=1:numallies
    neighborknowledge(alliedtracker(i),alliedtracker(i))=time;
    neighbors(i).partner=[];
end

newneighborknowledge=neighborknowledge;

for i=1:numallies
    counter=1;
    for j=1:numallies
        if i~=j
            if adjacency(i,j)==1
                neighbors(i).partner(counter)=alliedtracker(j);
                counter=counter+1;
            end
        end
    end
    if numel(neighbors(i).partner)==0
        newneighborknowledge(alliedtracker(i),:)=0;
        newneighborknowledge(alliedtracker(i),alliedtracker(i))=time;
    end
end

for i=1:numallies
    for j=1:numel(neighbors(i).partner)
            for k=1:numallies
                if
neighborknowledge(alliedtracker(i),alliedtracker(k))>neighborknowledge(
neighbors(i).partner(j),alliedtracker(k))

newneighborknowledge(neighbors(i).partner(j),alliedtracker(k))=neighbor
knowledge(alliedtracker(i),alliedtracker(k));
                end
            end
    end
end

dumsize=length(newneighborknowledge);
```

```
for i=1:dumsize
    for j=1:dumsize
        if i~=j
            if newneighborknowledge(i,j)-oldneighborknowledge(i,j)==1;
            else
                if oldneighborknowledge(i,j)~=0
                    newneighborknowledge(i,j)=0;
                end
            end
        end
    end
end

activeenemies(1:numenemies)=0;
end
```

Appendix C.7 CONDENSEOUTPUT.M

```
function out =
condenseoutput(out,outputmetrics,metrics,numiter,runnumber)

uncovered=zeros(numiter,3);

timeworsethannocom=zeros(1,3);
uncovered=timeworsethannocom;
sumresource=uncovered;
timeatworst=uncovered;
timeatbest=uncovered;
totalenemies=0;

for i=1:numiter
    for j=1:3
        if j~=2
        best=outputmetrics(i,5+j);
        worst=outputmetrics(i,10+j);
        actual=outputmetrics(i,j);

        if best>worst
        normalized(i,j)=(actual-worst)/(best-worst);
        elseif best<=worst
            normalized(i,j)=.5;
        else
            normalized(i,j)=0;
        end
        sumresource(j)=sumresource(j)+normalized(i,j);
        if normalized(i,j)<0
            timeworsethannocom(j)=timeworsethannocom(j)+1;
        end
        if normalized(i,j)==0
            timeatworst(j)=timeatworst(j)+1;
        elseif normalized(i,j)>=1
            timeatbest(j)=timeatbest(j)+1;
            normalized(i,j)=1;
        end
        end
    end
    uncovered(1)=uncovered(1)+outputmetrics(i,4);
    uncovered(2)=uncovered(2)+outputmetrics(i,9);
    totalenemies=totalenemies+outputmetrics(i,10);
end

uncovered(3)=uncovered(1)/totalenemies;
```

```
average(1:3)=sumresource(1:3)/numiter;

%Store information
out(runnumber,1:3)=average;
out(runnumber,4:6)=uncovered;
out(runnumber,7:9)=timeworsethannocom/numiter;
out(runnumber,10:12)=timeatworst/numiter;
out(runnumber,13:15)=timeatbest/numiter;
```

Appendix C.8 CONTROLASSIGNMENT.M

```
function
[assignment,perfectassignment,nocomassignment,truecost,activeenemies,en
emytimecover,eachallyenemytimeleft]=controlassignment(cost,localinfo,st
rategy,costchangedetected,oldassignment,enemytime,enemytimecover,eachal
lyenemytimeleft)

[numallies,numenemies]=size(cost);

perfectcost=cost;
nocomcost=cost;

counter=1;
activeenemies=0;

for j=1:numenemies
    if enemytimecover(j)==1
        enemytimecover(j)=enemytime(j);
        activeenemies(counter)=j;
        counter=counter+1;
    elseif enemytimecover(j)>1
        enemytimecover(j)=enemytimecover(j)-1;
        for i=1:numallies
            localinfo(i).cost(:,j)=0;
        end
        perfectcost(:,j)=0;
    end
end

for i=1:numallies
    for j=1:numenemies
        if eachallyenemytimeleft(i,j)==1
            eachallyenemytimeleft(i,j)=enemytime(j);
        elseif eachallyenemytimeleft(i,j)>1
            eachallyenemytimeleft(i,j)=eachallyenemytimeleft(i,j)-1;
            nocomcost(i,j)=0;
        end
    end
end

truecost=perfectcost;

zerotracker=sum(perfectcost,1);
for j=length(zerotracker):-1:1
    if zerotracker(j)==0
```

```
        perfectcost(:,j)=[];
        for i=1:numallies
            localinfo(i).cost(:,j)=[];
        end
    end
end

for i=1:numallies
    [localallies,localenemies]=size(localinfo(i).cost);
    minredun=ceil(localenemies/localallies);
    strategy.node(i)=updateredunstrategy(minredun,strategy.node(i));
end

[nocomassignment,distribution]=nocomcase(nocomcost,1);

assignment=floor(cost);
perfectassignment=assignment;

if activeenemies(1)>0
    preperfectassignment=idealmatching(perfectcost,1,1);

[preassignment]=normalmatching(localinfo,perfectcost,strategy,costchang
edetected,oldassignment);
    counter=1;
    for i=1:numenemies
        if sum(ismember(activeenemies,i))>=1
            assignment(:,i)=preassignment(:,counter);
            perfectassignment(:,i)=preperfectassignment(:,counter);
            counter=counter+1;
        end
    end
end

end
```

Appendix C.9 CONTROLTEST.M

```
function
[overalloutput,extrainfo,infocount]=ControlTest(inputs,overalloutput,it
erationnumber,runnumber,extrainfo,infocount)

close all
numiter=50;
numinitialunits = inputs(1);
numinitialenemies = inputs(2);
nettype = inputs(8);
nettype2 = inputs(9);
netvar2=inputs(10);
netvar3=inputs(11);
addremovesetinterval = inputs(3);
addremoveenemysetinterval  =  inputs(4);        %amount  of  time  between
addition/removal of enemies
modifylinkssetinterval  =  inputs(5);           %amount  of  time  between
link modification
changecostsetinterval = inputs(6);
networktype=[nettype,nettype2];
networkproperties=[numinitialunits,netvar2,netvar3];
addremoveflagadd = inputs(12);                           %dictates means
of addition/removal of units
addremoveflagremove  =  inputs(12);                             %Same  as
flagadd
addremoveflagtime = inputs(13);                         %
addremovewhichtoremoveflag = inputs(14);                %
addremoveminadd = inputs(15);                            %minimum amount
to be added
addremovemaxadd = inputs(16);                             %max amount to
be added
addremoveaddchance = inputs(17);                        %
addremoveremovechance  =  inputs(17);                           %same  as
addchance
addremovetotalmin = inputs(18);                                   %total
minimum number of units
addremovetotalmax = inputs(19);                                   %total
maximum number of units
addremovelinkmin = inputs(20);                                   %
addremovelinkmax = inputs(21);                                %
addremovenominal = numinitialunits;
addremoveenemyflagadd = inputs(22);                            %dictates
means of addition/removal of units
addremoveenemyflagtime = inputs(23);                       %
addremoveenemywhichtoremoveflag = inputs(24);              %
addremoveenemyminadd = inputs(25);                              %minimum
amount to be added
```

```
addremoveenemymaxadd = inputs(26);                           %max amount
to be added
addremoveenemychance = inputs(27);                         %
addremoveenemytotalmin = inputs(28);                              %total
minimum number of units
addremoveenemytotalmax = inputs(29);                              %total
maximum number of units
addremoveenemylinkmin = inputs(30);                        %
addremoveenemylinkmax = inputs(31);                        %
if addremovelinkmin>=addremovelinkmax
    addremovelinkmin=addremovelinkmax/2;
end
addremoveenemylinkaddflag = inputs(32);
modifylinksflagtime = inputs(33);            %
modifylinksaddflag = inputs(34);             %
modifylinksaddflagnumber = inputs(35);       %
modifylinksaddchance = inputs(36);          %
modifylinksminadd = inputs(37);              %
modifylinksmaxadd = inputs(38);              %
if modifylinksminadd>=modifylinksmaxadd
    modifylinksminadd=modifylinksmaxadd/2;
end
modifylinksremoveflag = inputs(39);          %
modifylinksremoveflagnumber = inputs(40);    %
modifylinksremovechance = modifylinksaddchance;       %
modifylinksminremoved = modifylinksminadd;          %
modifylinksmaxremoved = modifylinksmaxadd;          %
changecostflagtime = inputs(41);
changecostflagchange = inputs(42);
changecostminchange = inputs(43);
changecostmaxchange = inputs(44);
if changecostminchange>=changecostmaxchange
    changecostminchange=changecostmaxchange/2;
end
changecostflaghowtochange = inputs(45);
changecostamounttochangemin = inputs(46);
changecostamounttochangemax = inputs(47);
mincost=inputs(48);
maxcost=inputs(49);
changedetectedon=inputs(7);
dataremovefeatureon=0;
strategy=initializestrategy();
addremovemat=[addremoveflagadd,addremoveflagremove,addremovesetinterval
,addremoveflagtime,addremovewhichtoremoveflag,addremoveminadd,addremove
maxadd,addremoveaddchance,addremoveremovechance,addremovetotalmin,addre
movetotalmax,addremovelinkmin,addremovelinkmax,addremovenominal];
addremoveenemymat=[addremoveenemyflagadd,addremoveenemysetinterval,addr
emoveenemyflagtime,addremoveenemywhichtoremoveflag,addremoveenemyminadd
,addremoveenemymaxadd,addremoveenemychance,addremoveenemytotalmin,addre
moveenemytotalmax,addremoveenemylinkmin,addremoveenemylinkmax,addremove
enemylinkaddflag];
```

```
modifylinksmat=[modifylinksflagtime,modifylinkssetinterval,modifylinksa
ddflag,modifylinksaddflagnumber,modifylinksaddchance,modifylinksminadd,
modifylinksmaxadd,modifylinksremoveflag,modifylinksremoveflagnumber,mod
ifylinksremovechance,modifylinksminremoved,modifylinksmaxremoved];
changecostmat=[changecostflagtime,changecostflagchange,changecostsetint
erval,changecostminchange,changecostmaxchange,changecostflaghowtochange
,changecostamounttochangemin,changecostamounttochangemax,mincost,maxcos
t];
alliedtracker=(1:numinitialunits);
enemytracker=(1:numinitialenemies);
totalunits=ones(numinitialunits,1);
totalenemies=ones(numinitialenemies,1);
totalunitcount=numinitialunits;
enemymintimer=3;
enemymaxtimer=5;
onechance=1.0;
for i=1:numinitialenemies
    if rand<onechance
        enemytime(i)=1;
    else
        enemytime(i)=randinterval(enemymintimer,enemymaxtimer);
    end
    enemytimeleft(i)=1;
end
nummetrics=6;
%Initialize Metrics
metrics.raw = [];
metrics.normal = [];
metrics.linknormal = [];
localmetrics.node(1).raw=[];
localmetrics.node(1).normal=[];
localmetrics.node(1).linknormal=[];
metrics.raw=initializemetrics(metrics.raw,nummetrics,numiter);
metrics.normal=initializemetrics(metrics.normal,nummetrics,numiter);
metrics.linknormal=initializemetrics(metrics.linknormal,nummetrics,numi
ter);
localmetrics.node(1).raw=initializelocalmetrics(localmetrics.node(1).ra
w,nummetrics,numiter);
localmetrics.node(1).normal=initializelocalmetrics(localmetrics.node(1)
.raw,nummetrics,numiter);
localmetrics.node(1).linknormal=initializelocalmetrics(localmetrics.nod
e(1).linknormal,nummetrics,numiter);
idealperformance(1).totalapplied=0;
idealperformance(1).wastedresource=0;
idealperformance(1).minapplied=0;
idealperformance(1).uncovered=0;
nocomperformance=idealperformance;
actualperformance=idealperformance;
%Initialization
[adjacency,enemymat,junk]=intitializesystem(numinitialunits,numinitiale
nemies,networktype,networkproperties);
cost=initializecost(enemymat,mincost,maxcost);
```

```
nocomenemytimeleft=ceil(cost);
neighborknowledge=eye(numinitialunits);
newenemycheck=zeros(numinitialunits,1);
lastchange=zeros(1,numinitialunits);
costchangetracker=zeros(1,numinitialunits);
costchangedetected=zeros(1,numinitialunits);
oldassignment.node(1).assignment=enemymat;
oldassignment.node(2:numinitialunits)=oldassignment.node(1);

storedinfo(1).cost=cost;
storedinfo(1).alliedtracker=alliedtracker;

addremovetargettime=addremovesetinterval;
addremoveenemytargettime=addremoveenemysetinterval;
modifylinkstargettime=modifylinkssetinterval;
changecosttargettime=changecostsetinterval;

for i=1:numiter
    if sum(eq(size(cost),size(enemymat)))<2
        cost(end+1:length(enemymat),:)=0;
        nocomenemytimeleft(end+1:length(enemymat),:)=0;
    end
    iter=i;

[storedinfo,neighborknowledge]=storeinformation(cost,adjacency,storedin
fo,alliedtracker,totalunits,enemytracker,totalenemies,neighborknowledge
,i);

metrics=determinemetrics(adjacency,enemymat,metrics,cost,i,nummetrics);
    %Determine Local Network and Strategy

[localinfo,strategy,localmetrics,newenemycheck]=determinelocalnetwork(c
ost,neighborknowledge,storedinfo,i,localmetrics,nummetrics,alliedtracke
r,totalunits,strategy,newenemycheck,costchangetracker);
    %Determine Local Data

[localinfo]=determinelocaldata(cost,neighborknowledge,storedinfo,i,stra
tegy,nummetrics,alliedtracker,totalunits,localinfo);
    %Implement Control Scheme

[assignment,idealassignment,nocomassignment,truecost,activeenemies,enem
ytimeleft,nocomenemytimeleft]=controlassignment(cost,localinfo,strategy
,costchangedetected,oldassignment,enemytime,enemytimeleft,nocomenemytim
eleft);
    %Evaluate Performance Metrics

[idealperformance]=evaluateperformance(idealassignment,cost,idealperfor
mance,i,truecost,activeenemies);

[nocomperformance]=evaluateperformance(nocomassignment,cost,nocomperfor
mance,i,truecost,activeenemies);
```

```
[actualperformance]=evaluateperformance(assignment,cost,actualperforman
ce,i,truecost,activeenemies);
    %Communicate with neighbors

[lastchange,neighborknowledge,activeenemies]=communicationstage(adjacen
cy,lastchange,neighborknowledge,alliedtracker,i,cost);
    %visualizenetwork(adjacency,enemymat,metrics,numiter);

[adjacency,enemymat,addremovetargettime,cost,alliedtracker,totalunits,l
astchange,neighborknowledge,newenemycheck,nocomenemytimeleft]=addremove
units(addremovemat,adjacency,enemymat,i,addremovetargettime,cost,mincos
t,maxcost,alliedtracker,totalunits,newenemycheck,lastchange,neighborkno
wledge,enemytime,nocomenemytimeleft);

[enemymat,addremoveenemytargettime,cost,newenemycheck,enemytracker,tota
lenemies,enemytime,enemytimeleft,nocomenemytimeleft]=addremoveenemyunit
s(addremoveenemymat,adjacency,enemymat,i,addremoveenemytargettime,cost,
mincost,maxcost,alliedtracker,totalunits,newenemycheck,enemytracker,tot
alenemies,enemytime,enemytimeleft,nocomenemytimeleft,enemymintimer,enem
ymaxtimer,onechance);
    [adjacency,modifylinkstargettime]                                  =
modifylinks(modifylinksmat,adjacency,i,modifylinkstargettime);
    %Update or Change Cost
    oldcost=cost;

[cost,changecosttargettime]=changecost(changecostmat,cost,changecosttar
gettime,i);

[costchangetracker,costchangedetected,oldassignment]=costchangeamount(c
ost,oldcost,assignment);
end

metrics=determinemetrics(adjacency,enemymat,metrics,cost,numiter+1,numm
etrics);
% visualizenetwork(adjacency,enemymat,metrics,numiter);
sheet=['Sheet' num2str(iterationnumber)];
name=[num2str(runnumber) ' Output ALL'];
outputmetrics=convertperformance(idealperformance,nocomperformance,actu
alperformance,i,storedinfo);
%Convert to condensed usefull output metrics
overalloutput=condenseoutput(overalloutput,outputmetrics,metrics,numite
r,iterationnumber);
%Outputmetrics to excel
xlswrite(name,outputmetrics,sheet);

end
```

Appendix C.10 CONVERTPERFORMANCE.M

```
function
outblock=convertperformance(ideal,nocom,actual,time,storedinformation)

outblock=zeros(time,15);

for i=1:time
    outblock(i,1)=actual(i).totalapplied;
    outblock(i,2)=actual(i).wastedresource;
    outblock(i,3)=actual(i).minapplied;
    outblock(i,4)=actual(i).uncovered;
    outblock(i,5)=storedinformation(i).numallies;
    outblock(i,6)=ideal(i).totalapplied;
    outblock(i,7)=ideal(i).wastedresource;
    outblock(i,8)=ideal(i).minapplied;
    outblock(i,9)=ideal(i).uncovered;
    outblock(i,10)=storedinformation(i).numenemies;
    outblock(i,11)=nocom(i).totalapplied;
    outblock(i,12)=nocom(i).wastedresource;
    outblock(i,13)=nocom(i).minapplied;
    outblock(i,14)=nocom(i).uncovered;
    outblock(i,15)=0;
end

end
```

Appendix C.11 CONVERTCOSTTOF.M

```
function f=convertcosttof(cost)

[numallies,numenemies]=size(cost);

f=0;

if numallies>numenemies
    cost(:,numenemies+1:numallies)=-500;
    for j=1:numallies
        f(1+(j-1)*numallies:j*numallies)=cost(j,:);
    end
else
    for j=1:numallies
        f(1+(j-1)*numenemies:j*numenemies)=cost(j,:);
    end
end
if size(f)<1

end
f=-f;

end
```

Appendix C.12 CONVERTMATCHINGTOASSIGNMENT.M

```
function
assignment=convertmatchingtoassignment(matching,strategy,numallies,nume
nemies)

[dummy1,dummy2]=size(matching);

assignment=zeros(numallies,numenemies);
counter=zeros(1,numallies);

for i=1:dummy1
    for j=1:dummy2
        if matching(i,j)~=0
            counter(j)=counter(j)+1;
        end
    end
end

assignmentmatrix=zeros(max(counter),numallies);
assignmentmatrix(1,:)=1;

if numel(strategy)==1
    matching=floor(matching);
    for i=1:numallies
        for j=1:counter(i)-1
            assignmentmatrix(j+1,i)=assignmentmatrix(j,i)*strategy;
        end
    end

    if max(counter)>1
        summat=sum(assignmentmatrix);
    else
        summat=assignmentmatrix;
    end

    for i=1:numallies
        assignmentmatrix(:,i)=assignmentmatrix(:,i)./summat(i);
    end
end

counter=ones(numallies,1);

for i=1:dummy2
    for j=1:dummy1
        if matching(j,i)~=0
```

```
            assignment(i,matching(j,i))=assignmentmatrix(counter(i),i);
            counter(i)=counter(i)+1;
        end
    end
end
```

Appendix C.13 COSTCHANGEAMOUNT.M

```
function [cost,targettime] = changecost(mat,cost,targettime,time)

flagtime = mat(1);
flagchange = mat(2);
setinterval = mat(3);
minchange = mat(4); %Either % or set amount
maxchange =  mat(5);
flaghowtochange = mat(6);
amounttochangemin = mat(7); %Percentage or amount
amounttochangemax = mat(8);
minvalue = mat(9);
maxvalue = mat(10);

[numallies,numenemies]=size(cost);

if time>=targettime
    if flagtime == 1
        interval=setinterval;
    elseif flagtime == 2
        interval=randinterval(intlower,intupper);
    end

    targettime = time+interval;

    nonzerocost=0;
    whichnonzero(1).located(1:2)=0;
    for i=1:numallies
        for j=1:numenemies
            if cost(i,j)>0
                nonzerocost=nonzerocost+1;
                whichnonzero(nonzerocost).locate(1)=i;
                whichnonzero(nonzerocost).locate(2)=j;
            end
        end
    end

    numnonzero=nonzerocost;

    %Determine Amount to change
    if flagchange==1
        minc=minchange;
        maxc=maxchange;
    elseif flagchange==2
        minc=floor(minchange*nonzerocost);
        maxc=ceil(maxchange*nonzerocost);
    end

    changeamount=rand*(maxc-minc)+minc;

    changeamount=round(changeamount*nonzerocost);

    if changeamount>nonzerocost
        changeamount=nonzerocost;
    end

    %Determine Which to Change
    dummy=randperm(numnonzero);
    tochange=dummy(1:changeamount);

    %Determine How Much to change
    if flaghowtochange==1 %Set amount
        for i=1:changeamount
            changeies=rand()*(amounttochangemax-
amounttochangemin)+amounttochangemin;
            dummy=rand();
            costi=whichnonzero(tochange(i)).locate(1);
            costj=whichnonzero(tochange(i)).locate(2);
            coststore=cost(costi,costj);
            if dummy<=.5
                coststore=coststore-changeies;
            else
                coststore=coststore+changeies;
            end
            if coststore>maxvalue
                coststore=maxvalue;
            elseif coststore<minvalue
                coststore=minvalue;
            end
            cost(costi,costj)=coststore;
        end
    elseif flaghowtochange==2
        for i=1:changeamount
            dummy=rand();
            costi=whichnonzero(tochange(i)).locate(1);
            costj=whichnonzero(tochange(i)).locate(2);
            coststore=cost(costi,costj);
            min=amounttochangemin*coststore;
            max=amounttochangemax*coststore;
            changeies=rand()*(max-min)+min;
            if dummy<=.5
                coststore=coststore-changeies;
            else
                coststore=coststore+changeies;
            end
            if coststore>maxvalue
```

```
                coststore=maxvalue;
            elseif coststore<minvalue
                coststore=minvalue;
            end
            cost(costi,costj)=coststore;
        end
    end
end
end
```

Appendix C.14 CREATENETWORK.M

```
function adj=createnetwork(pointer,array)

if pointer(1)~=1
    array(2)=round(array(2)*array(1));
end

if pointer(1) == 1 %Erdos-Renyi

    a = array(1); %number of nodes
    b = array(2); %number of links OR probability of link
    adj = eye(a);

    if pointer(2) == 1 %Set number of links
        if b >= (a*(a-1)/2)
            adj = ones(a);
        else
            dummy = randperm(a*(a-1)/2);
            dummy = dummy(1:b);
            dummy = sort(dummy);

            flag = 1;
            count = 1;
            for i=1:a-1
                for j=i+1:a
                    if count==dummy(flag)
                        adj(i,j)=1;
                        adj(j,i)=1;
                        flag=flag+1;
                    end
                    count=count+1;
                    if flag > b
                        break
                    end
                end
                if flag>b
                    break
                end
            end

        end
    else %set probability of links
        for i=1:a-1
            for j=i+1:a
                if rand()<b
```

```
                    adj(i,j)=1;
                    adj(j,i)=1;
                end
            end
        end
    end

elseif pointer(1) == 2 % Watts-Strogatz
    a = array(1); %number of nodes
    b = array(2); %number of neighbors
    c = array(3); %probability of changes

    adj = eye(a);

    if b>=a
        adj=ones(a);
    else

        for i=1:a
            for j=1:b/2
                if i-j>0
                    adj(i,i-j)=1;
                else
                    adj(i,a+i-j)=1;
                end
                if i+j<a+1
                    adj(i,i+j)=1;
                else
                    adj(i,i+j-a)=1;
                end
            end
        end

        if pointer(2) == 1 %Rewiring
            for i=1:a-1
                for j=i+1:a
                    if rand()<c %rewire
                        dummy = randperm(a);
                        dummy = dummy(1:2);

                        adj(dummy(1),dummy(2))=1; %add new link
                        adj(dummy(2),dummy(1))=1; %add new link

                        adj(i,j)  = 0; %remove old link
                        adj(j,i)  = 0; %remove old link

                    end
                end
            end
        else %Adding Links
```

```
                for i=1:a-1
                    for j=i+1:a
                        if rand()<c %rewire
                            dummy = randperm(a);
                            dummy = dummy(1:2);

                            adj(dummy(1),dummy(2))=1; %add new link
                            adj(dummy(2),dummy(1))=1; %add new link
                        end
                    end
                end
            end
        end
    elseif pointer(1) == 3 % Exponential
        a = array(1); %Number of nodes
        b = array(2); %Number of links per addition

        adj = eye(a);

        for i=2:b+1
            adj(i,1:i)=1;
            adj(1:i,i)=1;
        end

        for i=b+2:a
            dummy = randperm(i-1);
            dummy = dummy(1:b);

            for j=1:numel(dummy);
                adj(i,dummy(j)) = 1;
                adj(dummy(j),i) = 1;
            end
        end

    elseif pointer(1) == 4 % Barabasi-Albert

        a = array(1); %Number of nodes
        b = array(2); %Number of links per addition

        adj = eye(a);

        for i=2:b+1
            adj(i,1:i)=1;
            adj(1:i,i)=1;
        end



        for i=b+2:a
            degreemat = adj-eye(a);

            degree = sum(degreemat);

            totaldegree = sum(degree(1:i-1));
            totalprob = degree(1:i-1)/totaldegree;

            cdf=totalprob;

            for j=2:numel(totalprob)
                cdf(j)=totalprob(j)+cdf(j-1);
            end

            for j=1:b
                dummyrand = rand();
                for k=1:numel(cdf)
                    if dummyrand<=cdf(k)

                        adj(i,k)=1;
                        adj(k,i)=1;

                        dummy=k;
                        break
                    end
                end


                if b>1
                    totalprob(k)=0;
                    totalprob=totalprob/sum(totalprob);
                    cdf(1)=totalprob(1);
                    for j=2:numel(totalprob)
                        cdf(j)=totalprob(j)+cdf(j-1);
                    end
                end

            end
            degree = sum(adj);
        end

    end
```

Appendix C.15 DETERMINELOCALDATA.M

```
function
[localinfo]=determinelocaldata(cost,neighborknowledge,storedinfo,time,s
trategy,nummetrics,alliedtracker,totalunits,localinfo)

[numallies,numenemies]=size(cost);

for i=1:numallies
    neighborknowledge(alliedtracker(i),alliedtracker(i))=time;
    localinfo(i).cost=zeros(1,numenemies);
    if time==1
        localinfo(i).oldassignment=0;
    end
end

maxenemies=numenemies;
maxallies=numallies;

%Determine Local Cost
for i=1:numallies
    lookbackcost=strategy.node(i).lookbackcost;
    for j=1:numallies
        if i~=j
            if
neighborknowledge(alliedtracker(i),alliedtracker(j))>(time-
lookbackcost) && neighborknowledge(alliedtracker(i),alliedtracker(j))>0

curtime=neighborknowledge(alliedtracker(i),alliedtracker(j));
                allymarker=alliedtracker(j);

[size1,size2]=size(storedinfo(curtime).cost(allymarker,:));
                if eq(size2,maxenemies)==0
                    strategy.node(i).lookbackcost=1;
                else
                    if
sum(eq(storedinfo(time).enemytracker,storedinfo(curtime).enemytracker))
<length(storedinfo(time).enemytracker)
                        strategy.node(i).lookbackcost=1;
                    end
                end
                if size1>maxallies
                    maxallies=size1;
                end
            end
        end
    end
```

```
end

for i=1:numallies
    localinfo(i).cost=zeros(length(totalunits),maxenemies);

localinfo(i).cost(1,1:numenemies)=storedinfo(time).cost(alliedtracker(i
),1:numenemies);
    localinfo(i).location=i;
    counter=2;
    lookbackcost=strategy.node(i).lookbackcost;
    for j=1:numallies
        if i~=j
            if
neighborknowledge(alliedtracker(i),alliedtracker(j))>(time-
lookbackcost) && neighborknowledge(alliedtracker(i),alliedtracker(j))>0

curtime=neighborknowledge(alliedtracker(i),alliedtracker(j));
                allymarker=alliedtracker(j);

localinfo(i).cost(counter,:)=storedinfo(curtime).cost(allymarker,:);
                counter=counter+1;
            end
        end
    end
    zerotracker=sum(localinfo(i).cost,2);
    dummyc=0;
    for j=length(zerotracker):-1:1
        if zerotracker(j)==0
            localinfo(i).cost(j,:)=[];
            if j<=localinfo(i).location
                dummyc=dummyc+1;
            end
        end
    end
    localinfo(i).location=localinfo(i).location-dummyc;
end
```

Appendix C.16 DETERMINELOCALMETRICS.M

```
function
localmetrics=determinelocalmetrics(storedinfo,alliedtracker,numallies,t
ime,nummetrics,localmetrics)

allieddiameter=0;
totaldiameter=0;
alliedconnectivity=0;
totalconnectivity=0;

for i=1:numallies
    nodenum=i;
    %Network Metrics

    adjacency=storedinfo(nodenum).adjacency;

    numallies=length(adjacency);
    localmetrics(time).node(nodenum).raw = [];
    localmetrics(time).node(nodenum).normal = [];
    localmetrics(time).node(nodenum).linknormal = [];

localmetrics(time).node(nodenum).raw=initializelocalmetrics(localmetric
s(time).node(nodenum).raw,nummetrics);

localmetrics(time).node(nodenum).normal=initializelocalmetrics(localmet
rics(time).node(nodenum).normal,nummetrics);

localmetrics(time).node(nodenum).linknormal=initializelocalmetrics(loca
lmetrics(time).node(nodenum).linknormal,nummetrics);

    alliedadjacency=adjacency;

    allieddiameter=metricdiameter(alliedadjacency);


[alliedconnectivity,alliedcheeger]=metricconnectivity(alliedadjacency);

    numlinks=sum(sum(adjacency-eye(numallies)))/2;
    maxlinks=(numallies/2*(numallies-1));

    percentoflinks=numlinks/maxlinks;
```

```
rawlocalmetrics=[allieddiameter,alliedconnectivity(1:2),alliedcheeger(1
:2),numlinks];

    rawlocalmetrics;

    localmetrics(time).node(nodenum).raw.value(:)=rawlocalmetrics;

localmetrics(time).node(nodenum).normal.value(:)=rawlocalmetrics/numall
ies;

localmetrics(time).node(nodenum).linknormal.value(:)=rawlocalmetrics/ma
xlinks;

    if time==1

localmetrics(time).node(nodenum).raw=determinesublocalmetrics(localmetr
ics(time).node(nodenum).raw,[],time,nummetrics);

localmetrics(time).node(nodenum).normal=determinesublocalmetrics(localm
etrics(time).node(nodenum).normal,[],time,nummetrics);

localmetrics(time).node(nodenum).linknormal=determinesublocalmetrics(lo
calmetrics(time).node(nodenum).linknormal,[],time,nummetrics);
    else
        if length(localmetrics(time-1))<nodenum
            localmetrics(time-
1).node(nodenum)=createghostdata(localmetrics(time).node(nodenum));
        end


localmetrics(time).node(nodenum).raw=determinesublocalmetrics(localmetr
ics(time).node(nodenum).raw,localmetrics(time-
1).node(nodenum).raw,time,nummetrics);

localmetrics(time).node(nodenum).normal=determinesublocalmetrics(localm
etrics(time).node(nodenum).normal,localmetrics(time-
1).node(nodenum).normal,time,nummetrics);

localmetrics(time).node(nodenum).linknormal=determinesublocalmetrics(lo
calmetrics(time).node(nodenum).linknormal,localmetrics(time-
1).node(nodenum).linknormal,time,nummetrics);
    end
end

end
```

Appendix C.17 DETERMINELOCALNETWORK.M

```
function
[localinfo,strategy,localmetrics,newenemycheck]=determinelocalnetwork(c
ost,neighborknowledge,storedinfo,time,localmetrics,nummetrics,alliedtra
cker,totalunits,strategy,newenemycheck,costchangetracker)

[numallies,numenemies]=size(cost);

maxallies=numel(totalunits);


% Determine Local Adjacency
for i=1:numallies
    newmaxallies=maxallies;
    localinfo(i).adjacency = eye(maxallies);
    localinfo(i).adjacency(alliedtracker(i),alliedtracker(i))=1;
    if time==1

localinfo(i).adjacency(alliedtracker(i),:)=storedinfo(time).adjacency(a
lliedtracker(i),:);

localinfo(i).adjacency(:,alliedtracker(i))=storedinfo(time).adjacency(:
,alliedtracker(i));
    else
        dummyadj=eye(maxallies);
        sizer=length(storedinfo(time-1).adjacency);
        dummyadj(1:sizer,1:sizer)=storedinfo(time-1).adjacency;


localinfo(i).adjacency(alliedtracker(i),:)=dummyadj(alliedtracker(i),:)
;

localinfo(i).adjacency(:,alliedtracker(i))=dummyadj(:,alliedtracker(i))
;
    end
    for j=1:numallies
        if i~=j
            if neighborknowledge(alliedtracker(i),alliedtracker(j))>0

curtime=neighborknowledge(alliedtracker(i),alliedtracker(j));
                sizer=length(storedinfo(curtime).adjacency);

dummyadj(1:sizer,1:sizer)=storedinfo(curtime).adjacency;
```

```
localinfo(i).adjacency(alliedtracker(j),:)=dummyadj(alliedtracker(j),:)
;

localinfo(i).adjacency(:,alliedtracker(j))=dummyadj(:,alliedtracker(j))
;
            end
        end
    end
    zerotracker=sum(localinfo(i).adjacency);
    for j=maxallies:-1:1
        if zerotracker(j)<=1
            localinfo(i).adjacency(j,:)=[];
            localinfo(i).adjacency(:,j)=[];
        end
    end
    dummy=localinfo(i).adjacency;
    if length(dummy)<=1
        localinfo(i).adjacency=1;
        dummy;
    end

    if length(localinfo(i).adjacency)~=length(alliedtracker)

    end

end

localmetrics=determinelocalmetrics(localinfo,alliedtracker,numallies,ti
me,nummetrics,localmetrics);
%Determine Local Strategy

for i=1:numallies

strategy.node(i)=determinestrategy(length(localinfo(i).adjacency),numen
emies,localmetrics(time).node(i),strategy.node(i),newenemycheck(alliedt
racker(i)),costchangetracker(i),neighborknowledge(i,:));

end
newenemycheck(:)=0;

end
```

Appendix C.18 DETERMINEMETRICS.M

```
function
metrics=determinemetrics(adjacency,enemymat,metrics,cost,i,nummetrics,m
axunits)

allieddiameter=0;
totaldiameter=0;
alliedconnectivity=0;
totalconnectivity=0;

%Network Metrics

numallies=length(adjacency);
[dummy,numenemies]=size(enemymat);

alliedadjacency=adjacency;

allieddiameter=metricdiameter(alliedadjacency);

[alliedconnectivity,alliedcheeger]=metricconnectivity(alliedadjacency);

numlinks=sum(sum(adjacency-eye(numallies)))/2;
maxlinks=(numallies/2*(numallies-1));

percentoflinks=numlinks/maxlinks;

rawmetrics=[allieddiameter,alliedconnectivity(1:2),alliedcheeger(1:2),n
umlinks];

metrics.raw.value(i,:)=rawmetrics;
metrics.normal.value(i,:)=rawmetrics/numallies;
metrics.linknormal.value(i,:)=rawmetrics/maxlinks;

metrics.raw=determinesubmetrics(metrics.raw,i,nummetrics);
metrics.normal=determinesubmetrics(metrics.normal,i,nummetrics);
metrics.linknormal=determinesubmetrics(metrics.linknormal,i,nummetrics)
;

newcost=cost;
newcost(~newcost)=nan;
%Nodal Metrics
for j=1:numallies
    %Number of enemies linked (with percent)
    metrics.raw.nodal.enemylinks(i,j)=sum(enemymat(j,:));
    metrics.raw.nodal.Nenemylinks(i,j)=sum(enemymat(j,:))/numenemies;
    %Average cost to enemy (max, min)
    metrics.raw.nodal.maxcost(i,j)=max(cost(j,:));
    metrics.raw.nodal.mincost(i,j)=min(newcost(j,:));

metrics.raw.nodal.avgcost(i,j)=sum(cost(j,:))/metrics.raw.nodal.enemyli
nks(i,j);
    %Number of allies linked (with percent)
    metrics.raw.nodal.alliedlinks(i,j)=sum(adjacency(j,:))-1;
    %Local connectivity
    metrics.raw.nodal.connectivity(i,j)=determinelocalcon(adjacency,j);

end

end
```

Appendix C.19 DETERMINESTRATEGY.M

```
function strategy =
determinestrategy(numallies,numenemies,metrics,strategy,newenemycheck,c
osttracker,neighbor)

diameter=metrics.raw.value(1);
normdiameter=metrics.normal.value(1);

diameterdiff=numallies-(diameter-1);
diameternorm=diameterdiff/numallies;
connectivity=metrics.normal.value(2);
cheeger=metrics.raw.value(4);

diameterchange=metrics.normal.change.value(1);
connectivitychange=metrics.normal.change.value(2);
cheegerchange=metrics.raw.change.value(4);

%Strategy Cost Cooldown

change=[-diameterchange,connectivitychange,cheegerchange];

zerocount=0;
for i=1:length(neighbor)
    if neighbor(i)==0
        zerocount=zerocount+1;
    end
end

for i=1:length(neighbor)
    if neighbor(i)+diameter<max(neighbor)+1
        if neighbor(i)~=0
            zerocount=zerocount+1;
        end
    end
end

if length(neighbor)-zerocount<numallies
    strategy.lookbackcost=1;
    strategy.psuedolookback=diameter-1;
end

if newenemycheck>=1
    strategy.lookbackcost=1;
    strategy.psuedolookback=0;
    strategy.redundancy=1;
    strategy.division=1;
end

if costtracker==1
    strategy.lookbackcost=1;
    strategy.psuedolookback=0;
    strategy.redundancy=1;
    strategy.division=1;
end

if min(change)<0
    strategy.lookbackcost=1;
    strategy.psuedolookback=0;
    strategy.redundancy=1;
    strategy.division=1;
else
    dumchange=abs(change);
    if max(dumchange)>0
        strategy.psuedolookback=strategy.psuedolookback+1;
        %     if strategy.psuedolookback>=numallies
        if strategy.psuedolookback>=diameter+1
            strategy.lookbackcost=numallies;
        end
    elseif max(dumchange)==0
        strategy.psuedolookback=strategy.psuedolookback+1;
        %     if strategy.psuedolookback>=numallies
        if strategy.psuedolookback>=diameter+1
            strategy.lookbackcost=numallies;
        end
    end
end

if strategy.lookbackcost>1
    strategystorage=[0 0 0 0];
    %Connectivity Check
    if connectivity>.2
        strategystorage(1)=1;
    elseif connectivity>.1
        strategystorage(1)=2;
    elseif connectivity>.05
        strategystorage(1)=3;
    else
        strategystorage(1)=4;
    end

    %Diameter Check
    if diameter<=3
        strategystorage(2)=1;
    elseif diameter<=4
        strategystorage(2)=2;
    elseif diameter<=5
```

```
        strategystorage(2)=3;
    else
        strategystorage(2)=4;
    end

    %Cheeger Check
    if cheeger>.2
        strategystorage(4)=1;
    elseif cheeger>.15
        strategystorage(4)=2;
    elseif cheeger>.1
        strategystorage(4)=3;
    else
        strategystorage(4)=4;
    end

    %Implement Proper Go
    usedstrat=max(strategystorage);

    strategy.division=1;
    if usedstrat==1
        strategy.redundancy=2;
        strategy.division=.25;
    elseif usedstrat==1
        strategy.redundancy=2;
    elseif usedstrat==2
        strategy.redundancy=3;
    elseif usedstrat==3
        strategy.redundancy=4;
    elseif usedstrat==4
        strategy.lookbackcost=1;
    end
end

end
```

Appendix C.20 EVALUATEPERFORMANCE.M

```
function
performancemetrics=evaluateperformance(assignment,cost,performancemetri
cs,time,truecost,activeenemies)

dummy=size(cost);
numallies=dummy(1);
numenemies=dummy(2);
numuncovered=0;

effectiveappliedresource(1)=.1;
%Evaluate Effective Applied Resource
if sum(eq(size(assignment),size(cost)))<2
    [duma,dumb]=size(assignment);
    cost(:,dumb)=0;
end

appliedresource=assignment.*truecost;
numuncovered=0;

counter=0;
for i=1:numenemies
    if sum(ismember(activeenemies,i))==1
        counter=counter+1;
        effectiveappliedresource(counter)=max(appliedresource(:,i));

performancemetrics(time).effective(counter)=effectiveappliedresource(co
unter);
        if effectiveappliedresource(counter)==0
            numuncovered=numuncovered+1;
        end
    end
end

totalappliedresource=sum(effectiveappliedresource);

averageresource=totalappliedresource/counter;

wastedresource=sum(sum(appliedresource))-totalappliedresource;

minappliedresource=min(effectiveappliedresource);

performancemetrics(time).totalapplied=totalappliedresource;
performancemetrics(time).wastedresource=wastedresource;
performancemetrics(time).averageresource=averageresource;
performancemetrics(time).minapplied=minappliedresource;
performancemetrics(time).uncovered=numuncovered;

end
```

Appendix C.21 FIXCOST.M

```
function newcost = fixcost(cost)

a=max(max(cost));

newcost=cost;
cost(cost==0)=inf;

b=min(min(cost));

[numallies,numenemies]=size(cost);

for i=1:numallies
    for j=1:numenemies
        if cost(i,j)==inf;
            newcost(i,j)=-500;
        end
        if cost(i,j)==a
            newcost(i,j)=newcost(i,j)+2*1e-1+j*1e-3;
        elseif cost(i,j)==b
            newcost(i,j)=newcost(i,j)-2*1e-1-j*1e-3;
        end
    end
end

end
```

Appendix C.22 FIXCOSTNORMAL.M

```
function newcost = fixcostnormal(cost,loc)

[duma,dumb]=size(cost);

if duma>1
    %Create Localized Structure
    dummystruct=1:duma;

    if loc>1
        dummystruct(1)=loc;
        for i=2:loc
            dummystruct(i)=dummystruct(i)-1;
        end
    end

    a=max(max(cost));

    newcost=cost;
    cost(cost==0)=inf;

    b=min(min(cost));

    [numallies,numenemies]=size(cost);

    for i=1:numallies
        for j=1:numenemies
            if cost(i,j)==inf;
                newcost(i,j)=-500;
            end
            if cost(i,j)==a
                newcost(i,j)=newcost(i,j)+dummystruct(i)*2e-1+j*1e-3;
            elseif cost(i,j)==b
                newcost(i,j)=newcost(i,j)-dummystruct(i)*2e-1-j*1e-3;
            end
        end
    end

else
    newcost=cost;
end

end
```

Appendix C.23 IDEALMATCHING.M

```
function assignment = idealmatching(cost,levelofredundancy,strategy)

fakecostint=fixcost(cost);
dummy=size(cost);

numallies=dummy(1);
numenemies=dummy(2);

numrequired=(levelofredundancy*numenemies);

psuedoallies=numallies*(ceil(numrequired/numallies)-1);

numrotations=ceil(numrequired/numallies);

psuedolevelofredundancy=levelofredundancy;
if ceil(numenemies/numallies)>levelofredundancy
    psuedolevelofredundancy=ceil(numenemies/numallies);
end

options = optimset('Display', 'off');

fakecost=fakecostint;
if numallies>=numenemies
    subround=ceil(numallies/numenemies);
    const=numallies^2;
    LB=zeros(const,1);
    UB=ones(const,1);
    fakecost(:,numenemies+1:numallies)=-500;
    f=convertcosttof(fakecost);

    A = zeros(2*numallies,const);
    B = ones(2*numallies,1);

    for j=1:numallies
        for k=1:numallies
            A(j,numallies*(j-1)+k)=1;
            A(j+numallies,j+numallies*(k-1))=1;
        end
    end

    Aeq=[];
    Beq=ones(numallies,1);
    for j=1:numallies
        for k=1:numallies
            Aeq(j,numallies*(j-1)+k)=1;
        end
    end

    for i=1:numrotations
        if numallies==numenemies
        x=linprog(f,A,B,A,B,LB,UB,[],options);
        else
        x=linprog(f,A,B,A,B,LB,UB,[],options);
        end

[semiassignment,matching(i,:),assigned,unassigned]=assignmentfromx(x,nu
mallies,numenemies,fakecost);
        if subround>1
            subfakecost=fakecost;
            %IF THIS IS THE LAST ASSIGNMENT ROUND DO SPECIAL TASKS
            for j=2:subround
                %Zero out appropriate rows
                for k=1:numel(assigned)
                    subfakecost(assigned(k),:)=-500;
                end
                Aeq=[];
                Beq=[];
                f=convertcosttof(subfakecost);
                newx=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,matching(i,:),assigned,unassigned]=assignmentfromxremai
nder(newx,numallies,numenemies,matching(i,:),j,subfakecost);
            end

        end

        if numel(unassigned)>0
            for unass=1:numel(unassigned)
                [dum,maxdummy]=max(cost(unassigned(unass),:));
                matching(unassigned(unass))=maxdummy;
            end
        end

        matchinguse=floor(matching);

        for j=1:numallies
            if matchinguse(i,j)~=0
                fakecost(j,matchinguse(i,j))=-500;
            end
        end
        f=convertcosttof(fakecost);
    end
```

```
elseif numallies<numenemies
    const=numallies*numenemies;
    LB=zeros(const,1);
    UB=ones(const,1);
    cost;

[prematching,fakecost,psuedoallies,psuedoenemies]=preassignment(cost,le
velofredundancy);

    extra=psuedoenemies-psuedoallies;
    psuedo=ceil(extra/psuedoallies);
    subround=ceil(psuedoenemies/psuedoallies);

    f=convertcosttof(fakecost);

    A = zeros(numallies+numenemies,const);
    B = ones(numallies+numenemies,1);

    for j=1:numallies
        for k=1:numenemies
            A(j,numenemies*(j-1)+k)=1;
            A(k+numallies,k+numenemies*(j-1))=1;
        end
    end

    for j=1:numallies
        for k=1:numenemies
            Aeq(j,numenemies*(j-1)+k)=1;
        end
    end
    Beq=ones(numallies,1);

    for i=1:levelofredundancy
        dummy=(i-1)*subround;
        x=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,matching(dummy+1,:),assigned,unassigned]=assignmentfrom
x(x,numallies,numenemies,fakecost);

        subfakecost=fakecost;
        if subround>1
            for j=2:subround
                for k=1:numel(assigned)
                    subfakecost(:,assigned(k))=-500;
                end

                f=convertcosttof(subfakecost);
                newx=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);
```

```
[semiassignment,matching(dummy+j,:),assigned,unassigned]=assignmentfrom
x(newx,numallies,numenemies,subfakecost);
            end
        end

        matchinguse=floor(matching);
        for j=1:numallies
            for k=0:subround-1
                if matchinguse(i+k,j)~=0
                    fakecost(j,matchinguse(i+k,j))=-500;
                end
            end
        end
        f=convertcosttof(fakecost);

    end

    matching=[prematching;matching];
end

unassigned=0;
counter1=1;
counter2=1;
loopflag=0;
newfakecost=fakecostint;
for i=1:numenemies
    if sum(sum(ismember(matching,i)))==0
        unassigned(counter1)=i;
        counter1=counter1+1;
        loopflag=1;
    else
        assigned(counter2)=i;
        counter2=counter2+1;
    end
end

if unassigned(1)~=0
    for i=1:numel(assigned)
        newfakecost(:,assigned(i))=-500;
    end
end

loopcounter=1;
while loopflag==1
    [duma,dumb]=size(newfakecost);
    if duma>dumb
        newfakecost(:,dumb+1:duma)=-500;
    end

    f=convertcosttof(newfakecost);
    x=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,dummymatching(1,:),assigned,unassigned]=assignmentfromx
(x,numallies,numenemies,newfakecost);
    matching=[matching;dummymatching];
    unassigned=0;
    assigned=0;
    counter1=1;
    counter2=1;
    for i=1:numenemies
        if sum(sum(ismember(matching,i)))==0
            unassigned(counter1)=i;
            counter1=counter1+1;
        else
            assigned(counter2)=i;
            counter2=counter2+1;
        end
    end

    if sum(dummymatching)==0
        loopflag=0;
        break
    end

    if loopcounter>numenemies
        loopflag=0;
        break
    end

    if numel(assigned)>=numenemies
        loopflag=0;
        break
    end

    loopcounter=loopcounter+1;
end

%Convert Matching scheme into direct assignment method
assignment=convertmatchingtoassignment(matching,strategy,numallies,nume
nemies);

assignmentcheck=sum(assignment,2);

if max(sum(assignment,2))<1

end

if min(assignmentcheck)==0
    for i=1:numallies
        if assignmentcheck(i)==0
```

```
            if sum(cost(i,:))>0
            dummy(1:numenemies)=0;
            for j=1:length(cost(i,:));
                if cost(i,j)>0
                    dummy(j)=ceil(cost(1,j));
                end
            end
            fakeassignment(1,:)=dummy./sum(dummy);
            assignment(1,:)=fakeassignment(1,:);
            end
        end
    end
end

end
```

Appendix C.24 INITIALIZECOST.M

```
function cost=initializecost(enemymat,min,max)

[numallies,numenemies]=size(enemymat);

for i=1:numallies
    for j=1:numenemies
        if enemymat(i,j)==1
            cost(i,j)=(max-min)*rand()+min;
        end
    end
end

end
```

Appendix C.25 INITIALIZESYSTEM.M

```
function [adjacency,enemymat,supplementalinfo] =
intitializesystem(numallied,numenemy,networktype,networkproperties)

linkmin = 3;
linkmax = numallied;

networkinfo = [networktype];


adjacency=createnetwork(networkinfo,networkproperties);

alliedtracker=(1:numallied);
totalunits=ones(numallied,1);
newenemycheck=zeros(numallied,1);
enemytracker=(1:numenemy);
totalenemies=ones(numenemy,1);
enemytime=enemytracker;
enemytimeleft=enemytracker;
nocomenemytimeleft=ones(numallied,numenemy);

[enemymat,junk,junk,junk,junk,junk,junk,junk,junk]=addremoveenemyunits(
[1,1,1,1,numenemy,numenemy,1,0,numenemy,linkmin,linkmax,1],adjacency,[]
,1,1,[],0,0,alliedtracker,totalunits,newenemycheck,enemytracker,totalen
emies,enemytime,enemytimeleft,nocomenemytimeleft,1,1,1);

supplementalinfo = [0];

end
```

Appendix C.26 METRICCONNECTIVITY.M

```
function [connectivity,cheeger] = metricconnectivity(mat)

dummy = size(mat);
a = dummy(1);

if a==1
    connectivity(1:3)=1;
    cheeger(1,1:2)=0;
else if mat==eye(a);
        connectivity(1:3)=0;
        cheeger(1,1:2)=0;
    else

        [lap,normlap]=adjacencytolaplacian(mat);

        [evec,eval]=eig(lap);
        [normvec,normval]=eig(normlap);

        connectedcomponents=0;

        for i=1:a
            if eval(i,i)<1e-13
                connectedcomponents=connectedcomponents+1;
            else
                break
            end
        end

        connectivity(1)=eval(2,2);
        connectivity(2)=trace(mat^3)/trace(ones(a)^3);
        connectivity(3)=connectedcomponents;

        cheeger(1,1)=normval(2,2)/2;
        cheeger(1,2)=sqrt(2*normval(2,2));

    end

end
```

Appendix C.27 METRICDIAMETER.M

```
function [diameter,nodepair] = metricdiameter(mat)

dummy = size(mat);
a = dummy(1);

check = 0;
diameter = 1;

testmat=mat;

checkcount=1;

checkmat=eye(a);

for i=1:a
    for j=1:a
        if testmat(i,j) ~= 0
            checkmat(i,j)=1;
        end
    end
end

while check == 0
    if sum(sum(checkmat)) == a^2
        break
    else
        diameter=diameter+1;
    end

    testmat=testmat*mat;

    for i=1:a
        for j=1:a
            if testmat(i,j) ~= 0
                checkmat(i,j)=1;
            end
        end
    end

    checkcount=checkcount+1;

    if checkcount > a
        diameter=a^2;
        break
    end

end


end
```

Appendix C.28 MODIFYLINKS.M

```
function [newadjacency,targettime] =
modifylinks(mat,adjacency,time,targettime)

numunits=length(adjacency);

maxlinks=numunits*(numunits-1)/2;
numlinks=sum(sum(adjacency-eye(numunits)))/2;
numopen=maxlinks-numlinks;

linkslist=zeros(numlinks,2);
openlist=zeros(numopen,2);

flagtime=mat(1);
setinterval=mat(2);
addflag=mat(3);
addflagnumber=mat(4);
addchance=mat(5);
minadd=mat(6);
maxadd=mat(7);
removeflag=mat(8);
removeflagnumber=mat(9);
removechance=mat(10);
minremoved=mat(11);
maxremoved=mat(12);

minpercent=minremoved;
maxpercent=maxremoved;

newadjacency=adjacency;

if time>=targettime
    %set new target time
    if flagtime == 1
        interval=setinterval;
    elseif flagtime == 2
        interval=randinterval(intlower,intupper);
    end

    targettime = time+interval;

    numremoved=0;

    %remove links
    if rand<removechance
        if removeflagnumber==1
            numremoved=randinterval(minremoved,maxremoved);
        elseif removeflagnumber==2
            minremove=floor(minpercent*numlinks);
            maxremove=ceil(maxpercent*numlinks);
            numremoved=randinterval(minremove,maxremove);
        elseif removeflagnumber==3
            minremove=floor(minpercent*maxlinks);
            maxremove=ceil(maxpercent*maxlinks);
            numremoved=randinterval(minremove,maxremove);
        end
    end

    if numremoved>numlinks
        numremoved=numlinks;
    end

    dummycount=0;

    if numremoved>0
        for i=1:numunits-1
            for j=i+1:numunits
                if adjacency(i,j)==1
                    dummycount=dummycount+1;
                    linkslist(dummycount,1:2)=[i,j];
                end
            end
        end

        if removeflag==1
            removelist=randperm(dummycount);
        elseif  removeflag==2  %Remove  from  higher  linked  nodes  with
higher probability

removelist=preferentiallinkremoval(adjacency,numremoved,linkslist,1);
        elseif  removeflag==3  %Remove  from  lower  linked  nodes  with
higher probability

removelist=preferentiallinkremoval(adjacency,numremoved,linkslist,2);
        end

        if length(removelist)<numremoved
            numremoved=length(removelist);
        end

        for i=1:numremoved

newadjacency(linkslist(removelist(i),1),linkslist(removelist(i),2))=0;

newadjacency(linkslist(removelist(i),2),linkslist(removelist(i),1))=0;
```

```
        end

    end

    %Add links

    numadded=0;

    if rand<addchance
        if addflagnumber==1
            numadded=randinterval(minadd,maxadd);
        elseif addflagnumber==2
            minadd=floor(minpercent*numlinks);
            maxadd=ceil(maxpercent*numlinks);
            numadded=randinterval(minadd,maxadd);
        elseif addflagnumber==3
            minadd=floor(minpercent*maxlinks);
            maxadd=ceil(maxpercent*maxlinks);
            numadded=randinterval(minadd,maxadd);
        end
    end

    if numadded+numlinks>maxlinks
        numadded=maxlinks-numlinks;
    end

    dummycount=0;

    if numadded>0
        for i=1:numunits-1
            for j=i+1:numunits
                if adjacency(i,j)==0
                    dummycount=dummycount+1;
                    openlist(dummycount,1:2)=[i,j];
                end
            end
        end

        if addflag==1
            addlist=randperm(dummycount);
        elseif addflag==2

addlist=preferentiallinkaddition(adjacency,numadded,openlist,2);
        elseif addflag==3

addlist=preferentiallinkaddition(adjacency,numadded,openlist,1);
        end
        for i=1:numadded

newadjacency(openlist(addlist(i),1),openlist(addlist(i),2))=1;
newadjacency(openlist(addlist(i),2),openlist(addlist(i),1))=1;
        end

    end

end

end
```

Appendix C.29 NOCOMCASE.M

```
function [assignment,distrib]=nocomcase(weight,strategy)

dummy=size(weight);

numallies=dummy(1);
numenemies=dummy(2);

numconnections=zeros(numallies,1);
connections=zeros(numallies,numenemies);
assignment=connections;
distrib=connections;

for i=1:numallies
    for j=1:numenemies
        if weight(i,j)~=0
            numconnections(i)=numconnections(i)+1;
            connections(i,j)=1;
        end
    end
end

if strategy==1 %Divides resource equally despite weights
    for i=1:numallies
        each=1/numconnections(i);
        assignment(i,:)=connections(i,:)*each;
        distrib(i,:)=weight(i,:)*each;
    end
elseif strategy==2 %Divides resources such that applied resource is
equal
    for i=1:numallies
        B=[];
        B(1,1)=1;
        B(2:numconnections(i)+1,1)=0;
        testmat=zeros(numconnections(i)+1);
        counter=1;
        testmat(1,:)=1;
        testmat(1,end)=0;
        for j=1:numenemies
            if weight(i,j)~=0
                counter=counter+1;
                testmat(counter,counter-1)=weight(i,j);
                testmat(counter,numconnections(i)+1)=-1;
                if counter>numconnections(i);
                    break
                end
            end
        end
        x=testmat\B;
        counter=1;
        for j=1:numenemies
            if connections(i,j)==1
                assignment(i,j)=x(counter);
                distrib(i,j)=x(end);
                if counter==numconnections(i)
                    break
                else
                    counter=counter+1;
                end
            end
        end
    end
end
```

Appendix C.30

NORMALCONVERTMATCHINGTOASSIGNMENT.M

```
function
assignment=normalconvertmatchingtoassignment(matching,strategy,numallie
s,numenemies)

[dummy1,dummy2]=size(matching);

assignment=zeros(numallies,numenemies);
counter=zeros(1,numallies);

for i=1:dummy1
    for j=1:dummy2
        if matching(i,j)~=0
            counter(j)=counter(j)+1;
        end
    end
end

assignmentmatrix=zeros(max(counter),numallies);
assignmentmatrix(1,:)=1;

if numel(strategy)==1
    matching=floor(matching);
    for i=1:numallies
        for j=1:counter(i)-1
            assignmentmatrix(j+1,i)=assignmentmatrix(j,i)*strategy;
        end
    end

    if max(counter)>1
        summat=sum(assignmentmatrix);
    else
        summat=assignmentmatrix;
    end

    for i=1:numallies
        assignmentmatrix(:,i)=assignmentmatrix(:,i)./summat(i);
    end
end

counter=ones(numallies,1);
for i=1:dummy2
```

```
    for j=1:dummy1
        if matching(j,i)~=0
            assignment(i,matching(j,i))=assignmentmatrix(counter(i),i);
            counter(i)=counter(i)+1;
        end
    end
end
```

Appendix C.31 NORMALMATCHING.M

```
function [assignment] =
normalmatching(localdata,cost,strategy,costchangedetected,oldassignment
)

[superallies,superenemies]=size(cost);

options = optimset('display', 'off');

assignment=zeros(superallies,superenemies);

for round=1:superallies

    newcost=localdata(round).cost;
    fakecost=fixcostnormal(newcost,localdata(round).location);

    [duma,dumb]=size(fakecost);
    if duma==0
        matching=1;

fakeassignment=normalconvertmatchingtoassignment(matching,1,1,superenem
ies);
    else

        levelofredundancy=strategy.node(round).redundancy;
        division=strategy.node(round).division;

        [numallies,numenemies]=size(newcost);


        numrequired=(levelofredundancy*numenemies);

        psuedoallies=numallies*(ceil(numrequired/numallies)-1);

        numrotations=ceil(numrequired/numallies);

        if numallies==1
            dummycost=ceil(newcost);
            matching=0;
            costcounter=1;
            for i=1:numenemies
                if dummycost(i)==1
                    matching(costcounter,1)=i;
                    costcounter=costcounter+1;
                end
            end
        else
            if numallies>=numenemies
                subround=ceil(numallies/numenemies);
                const=numallies^2;
                LB=zeros(const,1);
                UB=ones(const,1);
                matching=zeros(1,numallies);
                fakecost(:,numenemies+1:numallies)=-500;
                f=convertcosttof(fakecost);

                A = zeros(2*numallies,const);
                B = ones(2*numallies,1);

                for j=1:numallies
                    for k=1:numallies
                        A(j,numallies*(j-1)+k)=1;
                        A(j+numallies,j+numallies*(k-1))=1;
                    end
                end

                Aeq=[];
                Beq=[];

                for i=1:numrotations
                    if numallies==numenemies
                        x=linprog(f,A,B,A,B,LB,UB,[],options);
                    else
                        x=linprog(f,A,B,A,B,LB,UB,[],options);
                    end

[semiassignment,matching(i,:),assigned,unassigned]=assignmentfromx(x,nu
mallies,numenemies,fakecost);
                    if subround>1
                        subfakecost=fakecost;
                        %IF THIS IS THE LAST ASSIGNMENT ROUND DO
SPECIAL STUFF
                        for j=2:subround
                            %Zero out appropriate rows
                            for k=1:numel(assigned)
                                subfakecost(assigned(k),:)=-500;
                            end
                            f=convertcosttof(subfakecost);

newx=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);
                            matchingdum=matching;

[semiassignment,matching(i,:),assigned,unassigned]=assignmentfromxremai
nder(newx,numallies,numenemies,matching(i,:),j,subfakecost);
                        end
                    end

                    matchinguse=floor(matching);

                    for j=1:numallies
                        if matchinguse(i,j)~=0
                            fakecost(j,matchinguse(i,j))=-500;
                        end
                    end
                    f=convertcosttof(fakecost);
                end

                loopflag=0;
                if min(min(matching))==0
                    loopflag=1;
                    subfakecost=fakecost;
                    loopcount=0;
                end

                while loopflag==1
                    loopcount=loopcount+1;
                    assigncounter=1;
                    [duma,dumb]=size(matching);
                    for j=1:dumb
                        if matching(end,j)==0
                            unassigned(assigncounter)=j;
                            assigncounter=assigncounter+1;
                        else
                            subfakecost(j,:)=-500;
                        end
                    end

                    f=convertcosttof(fakecost);
                    x=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);
                    dummymatching=zeros(1,numallies);

[semiassignment,dummymatching,assigned,unassigned]=assignmentfromx(x,nu
mallies,numenemies,subfakecost);

                    for j=1:length(dummymatching(loopcount,:))
                        if        dummymatching(loopcount,j)>0        &&
matching(end,j)==0
                            matching(end,j)=dummymatching(loopcount,j);
                        end
                    end

                    if min(min(matching))>=0
                        loopflag=0;
                    end
                    if loopcount>numenemies
                        loopflag=0;
                        matching
                    end

                    if sum(dummymatching)==0
                        loopflag=0;
                        break
                    end

                end

            elseif numallies<numenemies
                const=numallies*numenemies;
                LB=zeros(const,1);
                UB=ones(const,1);
                matching=zeros(1,numallies);

[prematching,fakecost,psuedoallies,psuedoenemies]=preassignment(localda
ta(round).cost,levelofredundancy);

                extra=psuedoenemies-psuedoallies;
                psuedo=ceil(extra/psuedoallies);
                subround=ceil(psuedoenemies/psuedoallies);

                f=convertcosttof(fakecost);

                A = zeros(numallies+numenemies,const);
                B = ones(numallies+numenemies,1);

                for j=1:numallies
                    for k=1:numenemies
                        A(j,numenemies*(j-1)+k)=1;
                        A(k+numallies,k+numenemies*(j-1))=1;
                    end
                end

                Aeq=[];
                Beq=[];

                Beq=ones(numallies,1);

                for j=1:numallies
                    for k=1:numenemies
                        Aeq(j,numenemies*(j-1)+k)=1;
                    end
                end

                if sum(size(Aeq,1)==size(Beq,1))<1
```

```
                    poop=1;
                end

                for i=1:levelofredundancy
                    %                 for i=1:numrotations
                    dummy=(i-1)*subround;
                    x=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,matching(dummy+1,:),assigned,unassigned]=assignmentfrom
x(x,numallies,numenemies,fakecost);
                    subfakecost=fakecost;
                    if subround>1
                        for j=2:subround
                            for k=1:numel(assigned)
                                subfakecost(:,assigned(k))=-500;
                            end
                            f=convertcosttof(subfakecost);

newx=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,matching(dummy+j,:),assigned,unassigned]=assignmentfrom
x(newx,numallies,numenemies,subfakecost);
                        end
                    end

                    matchinguse=floor(matching);
                    for j=1:numallies
                        for k=0:subround-1
                            if matchinguse(i+k,j)~=0
                                fakecost(j,matchinguse(i+k,j))=-500;
                            end
                        end
                    end
                    f=convertcosttof(fakecost);

                end

                matching=[prematching;matching];
            end
            %         end

            %Assign Still Unassigned Allies
            unassigned=0;
            counter1=1;
            counter2=1;
            loopflag=0;
            newfakecost=newcost;
            for i=1:numenemies
                if sum(sum(ismember(matching,i)))==0
                    unassigned(counter1)=i;
                    counter1=counter1+1;
                    loopflag=1;
                else
                    assigned(counter2)=i;
                    counter2=counter2+1;
                end
            end

            if unassigned(1)~=0
                for i=1:numel(assigned)
                    newfakecost(:,assigned(i))=-500;
                end
            end

            loopcounter=1;
            while loopflag==1
                [duma,dumb]=size(newfakecost);
                if duma>dumb
                    newfakecost(:,dumb+1:duma)=-500;
                end

                f=convertcosttof(newfakecost);
                x=linprog(f,A,B,Aeq,Beq,LB,UB,[],options);

[semiassignment,dummymatching,assigned,unassigned]=assignmentfromx(x,nu
mallies,numenemies,newfakecost);
                matching=[matching;dummymatching];
                unassigned=0;
                assigned=0;
                counter1=1;
                counter2=1;
                for i=1:numenemies
                    if sum(sum(ismember(matching,i)))==0
                        unassigned(counter1)=i;
                        counter1=counter1+1;
                    else
                        assigned(counter2)=i;
                        counter2=counter2+1;
                    end
                end

                if sum(dummymatching)==0
                    loopflag=0;
                    break
                end

                if loopcounter>numenemies
                    loopflag=0;
                    break
                end

                if numel(assigned)>=numenemies
```

```
                    loopflag=0;
                    break
                end

                loopcounter=loopcounter+1;
            end
        end
        %Convert Matching scheme into direct assignment method

fakeassignment=normalconvertmatchingtoassignment(matching,division,numa
llies,numenemies);
    end
    [duma,dumb]=size(newcost);

        if sum(fakeassignment(1,:))==0
            dummy(1:numenemies)=0;
            for j=1:length(newcost(1,:));
                if newcost(1,j)>0
                    dummy(j)=ceil(newcost(1,j));
                end
            end
            fakeassignment(1,:)=dummy./sum(dummy);
        end

    assignment(round,1:superenemies)=fakeassignment(1,1:superenemies);
    if sum(assignment(round,1:superenemies))<.999

assignment(round,:)=assignment(round,:)/sum(assignment(round,:));
    end
end

end
```

Appendix C.32 PREASSIGNMENT.M

```
function [matching,fakecost,newallies,newenemies] =
preassignment(cost,levelofredundancy)

fakecost=cost;
dummy=size(cost);
numallies=dummy(1);
numenemies=dummy(2);
newallies=numallies;
newenemies=numenemies;

matchings=zeros(1,numallies);
matching=zeros(1,numallies);

preassigned=zeros(levelofredundancy,numenemies);
numassigned=zeros(1,numenemies);
matrix=zeros(numallies,numenemies);

for i=1:numallies
    for j=1:numenemies
        if fakecost(i,j)==-500
            fakecost(i,j)=0;
        end
    end
end

compatibility=ceil(fakecost);

if numallies>1
    numalliespossible=sum(compatibility);
else
    numalliespossible=compatibility;
end

minpossible=min(numalliespossible);

counter=ones(numenemies,1);

if minpossible<=levelofredundancy
    for i=minpossible:levelofredundancy
        for j=1:numenemies
            if numalliespossible(j)==i
                for k=1:numallies
                    if compatibility(k,j)==1
                        preassigned(counter(j),j)=k;
                        numassigned(1,j)=i;
                        counter(j)=counter(j)+1;
                    end
                end
            end
        end
    end
end

counter=ones(numenemies,1);

dum1=size(preassigned);
dum2=dum1(2);
dum1=dum1(1);

counter=ones(1,numallies);

for i=minpossible:levelofredundancy
    for j=1:numenemies
        if numassigned(j)==i
            for k=1:i
                dummy=preassigned(k,j);
                matching(counter(dummy),dummy)=j;
                counter(dummy)=counter(dummy)+1;
            end
        end
    end
end

assigned(1)=0;

for i=1:numenemies
    if sum(sum(ismember(matching,i)))>=1
        assigned(end+1)=i;
    end
end

assigned(1)=[];

if numel(assigned)>0
    for i=1:length(assigned)
        fakecost(:,assigned(i))=0;
        newenemies=newenemies-1;
    end
end

[a,b]=size(fakecost);
for i=1:a
    for j=1:b
        if fakecost(i,j)==0
```

```
            fakecost(i,j)=-500;
        end
    end
end


end
```

Appendix C.33 RUNCASES.M

```
clc
DoE=xlsread('Expanded Test','Sheet1');
numruns=size(DoE);
numruns=numruns(1);
intervalstore=[1 3 5 10 25 50];
numrepeats=5;
runnumber=1;
repeatnumber=1;
extrainfo=zeros(1,7);
infocount=0;
averageoutput=zeros(numruns,15);
for i=1:numruns
    i
    overalloutput=zeros(1,15);
    rawdata=DoE(i,:);
    dummydata=[0 0 0 0];
    dummydata=rawdata(3:6);
    for j=1:4
        rawdata(j+2)=intervalstore(dummydata(j));
    end
    for n=1:numrepeats
    n
[overalloutput,extrainfo,infocount]=ControlTest(rawdata,overalloutput,n
,i,extrainfo,infocount);
    end
    name=[num2str(i) ' Output Overall'];
    xlswrite(name,overalloutput,'Sheet1','A3');
    dummy=averageoutputmetrics(overalloutput,numrepeats);
    averageoutput(i,:)=dummy;
end

xlswrite('Average Output',averageoutput,'Sheet1','A3');
xlswrite('aUncovered',extrainfo,'Sheet1','A3');
```

Appendix C.34 STOREINFORMATION.M

```
function
[storedinfo,newneighborknowledge]=storeinformation(cost,adjacency,store
dinfo,alliedtracker,totalunits,enemytracker,totalenemies,neighborknowle
dge,time)

[numallies,numenemies]=size(cost);
totalunitcount=length(totalunits);

newneighborknowledge=neighborknowledge;

storedinfo(time).numallies=numallies;
storedinfo(time).numenemies=numenemies;
storedinfo(time).cost=zeros(totalunitcount,numenemies);
storedinfo(time).adjacency=zeros(totalunitcount);
storedinfo(time).alliedtracker=alliedtracker;
storedinfo(time).enemytracker=enemytracker;

counter=1;
for i=1:length(alliedtracker)
        keep=alliedtracker(i);
        storedinfo(time).cost(keep,:)=cost(i,:);
end

for i=1:length(alliedtracker)
    for j=1:length(alliedtracker)
        keepi=alliedtracker(i);
        keepj=alliedtracker(j);
        storedinfo(time).adjacency(keepi,keepj)=adjacency(i,j);
    end
end

for i=1:totalunitcount
    if totalunits(i)==0
        newneighborknowledge(:,i)=0;
    end
end
```

Appendix C.35 UPDATEREDUNSTRATEGY.M

```
function strategy = updateredunstrategy(minredun,strategy)

if minredun>4
    minredun=4;
end

if minredun==4
    strategy.lookbackcost=1;
    strategy.redundancy=1;
    strategy.division=1;
else
    if minredun>strategy.redundancy
        strategy.redundancy=minredun;
    end
end
```

Appendix C.36 VISUALIZENETWORK.M

```
function junk=visualizenetwork(adjacency,enemymat,metrics,numiter)

numallied=length(adjacency);
numenemy=size(enemymat);
numenemy=numenemy(2);

positionallied=zeros(numallied,2);
positionalliedall=zeros(numallied,2);
positionenemyall=zeros(numenemy,2);

for i=1:size((metrics.raw.value),1)
    x(i)=i;
    yuse(i)=i-1;
end

for i=1:numallied
    theta=2*pi*i/numallied+pi/2;
    positionallied(i,1)=cos(theta);
    positionallied(i,2)=sin(theta);
end

edgecountallied=0;

for i=1:numallied
    for j=i+1:numallied
        if adjacency(i,j)==1
            edgecountallied=edgecountallied+1;

            edgeallied(edgecountallied,1)=positionallied(i,1);
            edgeallied(edgecountallied,3)=positionallied(i,2);
            edgeallied(edgecountallied,2)=positionallied(j,1);
            edgeallied(edgecountallied,4)=positionallied(j,2);
        end
    end
end

positionalliedall(1:numallied,1)=-1;
positionenemyall(1:numenemy,1)=1;

if numallied>1
for i=1:numallied
    ypos=1-2.0/(numallied-1)*(i-1);
    positionalliedall(i,2)=ypos;
end
```

```
else
    positionalliedall(1,2)=1;
end

for i=1:numenemy
    ypos=1-2.0/(numenemy-1)*(i-1);
    positionenemyall(i,2)=ypos;
end

edgecountall=0;

for i=1:numallied
    for j=1:numenemy
        if enemymat(i,j)==1
            edgecountall=edgecountall+1;

            edgeall(edgecountall,1)=positionalliedall(i,1);
            edgeall(edgecountall,2)=positionenemyall(j,1);
            edgeall(edgecountall,3)=positionalliedall(i,2);
            edgeall(edgecountall,4)=positionenemyall(j,2);
        end
    end
end

subplot(2,2,1)
scatter(positionallied(:,1),positionallied(:,2),100,'filled','blue')
axis([-1.2,1.2,-1.2,1.2]);
axis off
for i=1:edgecountallied
    line([edgeallied(i,1:2)],[edgeallied(i,3:4)])
end

subplot(2,2,2)
scatter(positionalliedall(:,1),positionalliedall(:,2),100,'filled','blu
e')
hold on
scatter(positionenemyall(:,1),positionenemyall(:,2),100,'filled','red')
axis([-1.2,1.2,-1.2,1.2]);
for i=1:edgecountall
    line([edgeall(i,1:2)],[edgeall(i,3:4)])
end
axis off
hold off
subplot(2,2,3)
plot(yuse,metrics.normal.value(:,2))
xlim([0,numiter]);

pause(.5)

end
```

REFERENCES

[1] ALBERT, R, JEONG, H. and BARABÁSI, A.-L. “Error and attack tolerance of complex networks” *Nature*, vol. 406, July 2000, pp. 376-382.

[2] ALBERT, R. and BARABÁSI, A.-L. “Statistical mechanics of complex networks” *Reviews of Modern Physics*, vol. 74, Jan 2002, pp. 47-97.

[3] ALT, H., BLUM, N., MEHLHORN, K. and PAUL, M. “Computing a maximum cardinality matching in a bipartite graph in time O(n1.5 / log n)” *Information Processing Letters*, vol. 37, 1991, pp. 237-240.

[4] ANTSAKLIS, P. J. and KOUTSOUKOS, X. D. “On Hybrid Control of Complex Systems: A Survey” *International Conference on the Automation of Mixed Processes*, vol. 3, Mar. 1998, pp. 1023-1045.

[5] ARENAS, A., DIAZ-GUILERA, A., KURTHS, J., MORENA, Y. and ZHOU, C. “Synchronization in complex networks” *Physics Reports*, vol. 469, 2008, pp. 93-153.

[6] ARKIN, E. and HASSIN, R. “On local search for weighted packet problems” *Mathematics of Operations Research*, vol. 23, 1998, pp. 640-648.

[7] BALAKRISHNAN, R. and RANGANATHAN, K. “A Textbook of Graph Theory” *Universitext*, Springer New York, 2000.

[8] BARABÁSI, A.-L. and ALBERT, R. “Emergence of Scaling in Random Networks” *Science*, vol. 286, Oct 1999, pp. 509-512.

[9] BARAT, A., BARTHÉLEMY, M., PASTOR-SATORRAS, R. and VESPIGNANI, A. “The architecture of complex weighted networks” *Proceedings of the National Academy of Sciences*, vol. 101 no. 11, Mar 2004, pp. 3747-3752.

[10] BARR, R. S., GLOVER, F. and KLINGMAN, D. “A new alternating basis algorithm for semi assignment networks” *Computers and Mathematical Programming*, US Government Printing Office, Washington DC, 1978.

[11] BATTITI, R. and TECCHIOLLI, G. “The reactive tabu search” *Operations Research Society of America Journal on Computing*, vol. 6, 1994, pp. 126-140.

[12] BERGE, C. “Graphs” Elsevier Science Publishers, Amsterdam, 2nd edition, 1985.

[13] BOCCALETTI, S., LATORA, V., MORENO, Y., CHAVEZ, M. and HWANG, D.-U. “Complex networks: structure and dynamics” *Physics Reports*, vol. 424, Jan 2006, pp. 174-308.

[14] BOKHARI, S. H. “On the mapping problem” *IEEE Transactions on Computers*, vol. c-30, pp. 207-214, 1981.

[15] BORNHOLDT, S. and ROHLF, T. “Topological Evolution of Dynamical Networks: Global Criticality from Local Dynamics” *Physical Review Letters*, vol. 84 no. 26, pp. 6114-6117, June 2000.

[16] BOURGEOIS F. and LASSALLE, J. C. “Algorithm 415: Algorithm for the assignment problem (rectangular matrices)” *Communication of Association for Computing Machinery*, vol. 14, 1971, pp. 805-806.

[17] BOURGEOIS F. and LASSALLE, J. C. “An extension of the Munkres algorithm for the assignment problem to rectangular matrices” *Communication of Association for Computing Machinery*, vol. 14, 1971, pp. 802-804.

[18] BREIKREUTZ, B.-J., STARK, C. and TYERS, M. “Osprey: a network visualization system” *Genome Biology*, vol. 4 no. 3, Jan. 2003.

[19] BRUEMMER, D. J., FEW, D. A., BORING, R. L., MARBLE, J. L., WALTON, M. C. and NIELSEN, C. W. “Shared Understanding for Collaborative Control” *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, vol. 35 no. 4, July 2005, pp. 494-504.

[20] BURIAN, J. “Complex Systems Tutorial” Retrieved October 5 2011 from http://eldar.cz/cognition/complex/.

[21] BURKARD, R., DELL’AMICO, M. and MARTELLO, S. “Assignment Problems” Society for Industrial and Applied Mathematics, Philadelphia PA, 2009, pp 1-198.

[22] BURKHARD, R. and FRÖHLICH, K. “Some remarks on 3-dimensional assignment problems” *Methods of Operations Research*, vol. 36, 1980, pp. 31-36.

[23] BURKOV, V. N., RUBINSTEIN, M. I. and SOKOLOV, V. B. “Some problems in optimal allocation of large-volume memories” *Avtomatika I Telemekhanika*, vol. 9, 1969, pp. 83-91.

[24] BUSER, P. “A note on the isoperimetric constant” *Annales scientifiques de l'École Normale Supérieure Sér. 4*, vol. 15 no. 2, 1982, pp. 213–230.

[25] BUTTS, C. T., PETRESCU-PRAHOVA, M. and CROSS, B. R. "Responder Communication Networks in the World Trade Center Disaster: Implications for Modeling of Communication Within Emergency Settings" *Journal of Mathematical Sociology*, vol. 31, 2007, pp 121-147.

[26] CARON, G., HANSEN, P. and JAUMARD, B. "The assignment problem with seniority and job priority constraints" *Operations Research*, vol. 47, 1999, pp. 449-453.

[27] CELA, E. "The Quadratic Assignment Problem Theory and Algorithms" Kluwer Academic Publishers, Boston MA, 1998, pp 1-100.

[28] CHAN, D. Y. C., HUGHES, B. D., LEONG, A. and REED, W. J. "Stochastically Evolving Networks" *Physical Review*, vol. 68 066124, 2003.

[29] CHANDLER, P. R., PACHTER, M., NYGARD, K. E. and SWAROOP, D. "Cooperative Control for Target Classification" *Applied Optimization*, vol. 66, 2002, pp. 1-19.

[30] CHANDLER, P. R., PACHTER, M., SWAROOP, D., FOWLER, J. M., HOWLETT, J. K., RASMUSSEN, S., SCHUMACHER, C. and NYGARD, K. E. "Complexity in UAV Cooperative Control" *American Control Conference*, vol. 3, 2002, pp. 1831-1836.

[31] CHEGIREDDY, C. R. and HAMACHER, H. W. "Algorithms for finding the k-best perfect matchings" *Discrete Applied Mathematics*, vol. 18, 1987, pp. 155-165.

[32] CHEN, Z., WILSON, K. A., JIN, Y. HENDRIX, W. and SAMATOVA N. F. "Detecting and Tracking Community Dynamics in Evolutionary Networks" *IEEE International Conference on Data Mining Workshops*, 2010, pp 318-327.

[33] CHERIYAN, J., HAGERUP, T., and MEHLHORN, K. “Can a maxmimum flow be computed in O(nm)” *Automata, Languages, and Programming*, vol. 443, Springer, New York, 1990, pp. 235-248.

[34] CHUNG, F. R. K. “Spectral Graph Theory” *Regional Conference Series in Mathematics*, no. 92, American Mathematical Society, Providence RI, 2000.

[35] CLAUSET, A., TANNER, H. G., ABDULLAH, C. T. and BYRNE, R. H. “Controlling across complex networks – Emerging links between networks and control” *Annual Reviews in Control*, vol. 32, pp. 183-192, 2008.

[36] CORTES, J., MARTINEZ, S., KARATAS, T. and BULLO, F. “Coverage control for mobile sensing networks” *IEEE Transactions on Robotics and Automation*, vol. 20 no. 2, 2004, pp. 243-255.

[37] CRAMA, Y., OERLEMANS, A. and SPIEKSMA, F. “Approximate algorithms for three-dimensional assignment problems with triangle inequalitites” *European Journal of Operational Research*, vol. 60, 1996, pp. 273-279.

[38] CRIADO, R., FLORES, J., GONZALEZ-VASCO, M. I. and PELLO, J. “Choosing a leader on a complex network” *Journal of Computational and Applied Mathematics*, vol. 204, Dec. 2005, pp. 10-17.

[39] DELL’AMICO, M. and MARTELLO, S. “The k-cardinality assignment problem” *Discrete Applied Mathematics*, vol. 79, 1997, pp. 103-121.

[40] DERMAN, C. and KLEIN, M. “A note on the optimal depletion of inventory” *Management Science*, vol. 5, 1959, pp. 210-213.

[41] DHAMDHERE, K., RAVI, R. and SINGH, M. "On Two-Stage Stochastic Minimum Spanning Trees" *Integer Programming and Combinatorial Optimization*, vol. 3509, 2005, pp. 189-199.

[42] DOGOROTSEV, S. N. and MENDES, J. F. F. "Evolution of networks" *Advances in Physics*, vol. 51 no. 4, Oct. 2001, pp. 1079-1187.

[43] DUNBAR, W. B. and MURRAY, R. M. "Distributed receding horizon control for multi-vehicle formation stabilization". *Automatica*, vol 42 no. 4, 2006, pp. 549-558.

[44] EGERVAVY, E. "Matrixok kombinatorius tulajdonságairol" *Matematikai és Fizikai Lapok*, vol. 38, 1931, pp. 16-28.

[45] ERDÖS, P. and RÉNYI, A. "On random graphs I." *Publicationes Mathematicae (Debrecen)*, 1959, pp 290-297.

[46] ERDÖS, P. and RÉNYI, A. "On the evolution of random graphs" *Publications of the Mathematical Institute of Hungary Academy of Sciences*, vol. 5, 1960, pp. 17-61.

[47] FAX, J. A. and MURRAY, R. M. "Information Flow and Cooperative Control of Vehicle Formations" *IEEE Transactions on Automatic Control*, vol. 49 no. 9, Sept. 2004, pp. 1465-1476.

[48] FEIGE, U., JAIN, K., MAHDIAN, M. and MIRROKNI, V. "Robust Combinatorial Optimization with Exponential Scenarios" *Integer Programming and Combinatorial Optimization*, vol. 4513, 2007, pp. 439-453.

[49] FEO, T. and RESENDE, M. G. C. "Greedy randomized adaptive search procedures" *Journal of Global Optimization*, vol. 6, 1995, pp. 109-133.

[50] FLINT, M., POLUCARPOU, M. and FERNANDEZ-GAUCHERAND, E. "Cooperative Control for Multiple Autonomous UAV's Searching for Targets" *Proceedings of the IEEE Conference on Decision and Control*, vol. 41, 2002, pp. 2823-2828.

[51] FONG, T., THORPE, C. and BAUR C. "Multi-Robot Remote Driving With Collaborative Control" *IEEE Transactions on Industrial Electronics*, vol. 50 no. 4, Aug. 2003.

[52] FORSBERG, J. H., DELANY, R. M., ZHAO, Q., HARAKAS, G. and CHANDRAN, R. "Analyzing lanthanide-included shifts in the NMW spectra of lanthanide(III) complexes derived from 1,4,7,10-tetrakis(N,N-diethylacetamido)-1,4,7,10-te-traazacyclododecane" *Inorganic Chemistry*, vol. 34, 1994, pp. 3705-3715.

[53] FRIEZE, A. M. and YADEGAR, J. "On the quadratic assignment problem" *Discrete Applied Mathematics*, vol. 5, 1983, pp. 89-98.

[54] GEOFFRION, A. M. and GRAVES, G. W. "Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/LP approach" *Operations Research*, vol. 24, 1976, pp. 595-610.

[55] GLOVER, F. "Maximum matching in a convex bipartite graph" *Naval Research Logistics Quarterly*, vol. 14, 1967, pp. 313-316.

[56] GLOVER, F. "Tabu search – part I" *Operations Research Society of America Journal on Computing*, vol. 1, 1989, pp. 190-206.

[57] GLOVER, F. "Tabu search – part II" Operations Research Society of America *Journal on Computing*, vol. 2, 1989, pp. 4-32.

[58] GOLDBERG, K. and CHEN, B. "Collaborative Control of Robot Motion: Robustness to Error" *IEEE/RSJ International Conference on Robots and Systems*, Oct. 2001.

[59] GOODWIN, G. C., SILVA, E. I. and QUEVEDO, D. E. "A Brief Introduction to the Analysis and Design of Networked Control Systems" *Control and Decision Conference,* 2008, pp. 1-13.

[60] GRAPHSTREAM TEAM 'GraphSteam Library" 2010-2013. Retrieved October 8 2011 from http://graphstream-project.org/doc/.

[61] GRÖTSCHEL, M. "Discrete Mathematics in Manufacturing" *Proceedings of the International Conference on Industrial and Applied Mathematics*, vol. 2, 1992, pp. 119-145.

[62] HAHN, P. and GRANT, T. "Lower bounds for the quadratic assignment problem based upon a dual formulation" *Operations Research,* vol. 46, no. 6, 1998, pp. 912-922.

[63] HAUSMANN, D., KORTE, B. and JENKYNS, T. "Worst cast analysis of greedy type algorithms for independence systems" *Mathematical Programming*, vol. 12, 1980, pp. 120-131.

[64] HERMAN, I., MELANCON, G. and MARSHALL M. S. "Graph Visualization and Navigation in Information Visualization: A Survery" *IEEE Transactions on Visualization and Computer Graphics*, vol. 6 no. 1, 2000.

[65] HOLDER, A. "Navy personnel planning and the optimal partition" *Operations Research*, vol. 53, 2005, pp. 77-89.

[66] HOPCROFT, J. and KARP, R. M. "An n5/2 algorithm for maximum matchings in bipartite graphs" *Society for Industrial and Applied Mathematics Journal of Computing*, vol. 2, 1973, pp. 225-231.

[67] HORN, W. A. "Minimizing average flow time with parallel machines" *Operations Research*, vol. 21, 1973, pp. 846-847.

[68] HOVARESHTI, P. "Consensus Problems and the Effects of Graph Topology in Collaborative Control" *Unpublished PhD Dissertation*, 2009.

[69] HSIEH, M.-Y. A., CROWLEY, A., KUMAR, V. and TAYLOR C. J. "Towards the deployment of a mobile robot network with end-to-end performance guarantees" *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006, pp 2085-2090.

[70] HURKENS, C. and SCHRIJVER, A. "On the size of systems of sets ever t of which have an sdr, with application to the worst-case ratio of heuristics for packaging problems" *Society of Industrial and Applied Mathematics Journal on Discrete Mathematics,* vol. 2, 1989, pp. 68-72.

[71] THE IGRAPH PROJECT "The Igraph Library" 2005-2013. Retrieved November 1, 2011 from http://igraph.sourceforge.net/.

[72] JADBABAIE, A., LIN, J. and MORSE, A. S. "Coordination of groups of mobile autonomous agents using nearest neighbor rules" *IEEE Transactions on Automatic Control*, vol. 48, June 2003, pp. 988-1001.

[73] JENSON, H. J. "Emergence of network structure in models of collective evolution and evolutionary dynamics" *Proceedings of the Royal Society*, vol. 464, 2008, pp. 2207-2217.

[74] JIANG, Z.-P. “Decentralized Control for Large-Scale Nonlinear Systems: A Review of Recent Results” *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Algorithms and Applications*, vol. 11, 2004, pp. 537-552.

[75] JONKER, R. and VOLGENANT, T. “Improving the Hungarian assignment algorithm” *Operations Research Letters*, vol. 5, 1986, pp. 171-175.

[76] KARP, R. “Reducibility among combinatorial problems” *Complexty of Computer Computations,* Plenum Press, New York, 1972.

[77] KAUFMAN, L. and BROECKX, F. “An algorithm for the quadratic assignment problem using Benders’ decomposition” *European Journal of Operational Research, vol. 2*, 1978, pp. 204-211.

[78] KEELING, M. “The implications of network structure for epidemic dynamics” *Theoretical Population Biology*, vol. 67, 2005, pp. 1-8.

[79] KEELING, M. J. and EAMES, K. T. D. “Networks and epidemic models” *Journal of the Royal Society Interface*, vol. 2, 2005, pp. 295-307.

[80] KENNINGTON, J. L. and WANG, Z. “A shortest augmenting path algorithm for the semi assignment problem. *Operations Research*, vol. 40, 1992, pp. 178-187.

[81] KITANO, H., ASADA, M., KUNIYOSHI, Y., NODA, I. AND OSAWA, E. “RoboCup: The Robot World Cup Initiative”. *AGENTS ’97 Proceedings of the first international conference on Autonomous agents*, 1997, pp. 340-347.

[82] KÖNIG, D. “Theorie der Endlichen und Unendlichen Graphen” *Akademische Verlagsgesellschaft*, Leipzig, 1936

[83] KOOPMANS, T. C. and BECKMANN, M. J. “Assignment problems and the location of economic activities” *Econometrica*, vol. 25, 1957, pp. 53-76.

[84] KRARUP, J. and PRUZAN, P. M. “Computer-aided layout design” *Mathematical Programming Study*, vol. 9, 1978, pp. 75-94.

[85] KRAUSE, W., GLAUCHE, I., SOLLACHER, R. and GREINER, M. “Impact of network structure on the capacity of wireless multihop ad hoc communication” *Physica A*, vol. 338, 2004, pp. 633-658.

[86] KUHN, H. W. “On combinatorial properties of matrices” *Office Naval Research Logistic Project Report*, 1955.

[87] KUHN, H. W. “On the origin of the Hungarian method” *History of Mathematical Programming*, North-Holland, Amsterdam, 1991, pp. 77-91.

[88] LAGUNA, M. “Methods and Strategies for Robust Combinatorial Optimization” *Operations Research Proceedings*, 1995.

[89] LAWLER, E. L. and WOOD, D. E. “Branch-and-Bound Methods: A Survey” *Operations Research*, vol. 14 no. 4, 1966, pp. 699-719.

[90] LEMMON, M. D. and ANTSAKLIS, P. J. “Timed Automata and Robust Control: Can We Now Control Complex Dynamical Systems?” *Proceedings of the 36th IEEE Conference on Decision and Control*, 1997, pp. 108-113.

[91] LERMAN, K. and GALSTYAN, A. “Mathematical Model of Foraging in a Group of Robots: Effect of Interference” *Autonomous Robots*, vol. 13, 2002, pp. 127-141.

[92] LI, L. “Topologies of Complex Networks: Functions and Structures” *Unpublished PhD thesis*, California Institute of Technology, 2007.

[93] LI, Y., PARDALOS, P. M. and RESENDE, M. “A greedy randomized search procedure for the quadratic assignment problem” *Assignment and Related Problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence RI, vol. 16, 1994, pp 237-261.

[94] MAGOS, D. and MILIOTIS, P. “An algorithm for the planar three-index assignment problem” *European Journal of Operations Research*, vol. 77, 1994, pp. 171-153.

[95] MCLAIN, T. W. and BEARD, R. W. “Coordination variables, coordination functions, and cooperative timing missions” Journal of Guidance, Control and Dynamics, vol. 28, Jan. 2005, pp. 150-161.

[96] MESBAHI, M. “On a Dynamic Extension of the Theory of Graphs” *Proceedings of the American Control Conference*, 2002, pp. 1234-1239.

[97] MILGRAM, S. “The Small-World Problem” *Psychology Today,* vol. 1 no. 1, May 1967, pp. 61-67.

[98] MILO, R., SHEN-ORR, S., ITZKOVITZ, S., KASHTAN, N., CHKLOVSKII, D. and ALON, U. “Network Motifs: Simple Building Blocks of Complex Networks” *Science*, vol. 298, Oct. 2002, pp. 824-827.

[99] MORRISON, T. “A New Paradigm for Robust Combinatorial Optimization: Using Persistence as a Theory of Evidence” *Unpublished PhD thesis*, 2010.

[100] MURRAY, R. M. “Recent Research in Cooperative Control of Multi-Vehicles Systems” *Dynamic Systems Measurement and Control-Transactions of the ASME*, vol. 129, 2007, pp. 571-583.

[101] MURTY, K. G. “An algorithm for ranking all the assignments in order of increasing cost” *Operations Research*, vol. 16, 1968, pp. 682-687.

[102] NEWMAN, M. E. J. “Assortative mixing in networks” *Physical Review Letters*, vol. 89 no. 20 208701, 2002.

[103] NEWMAN, M. E. J. “Modularity and community structure in networks” *Proceedings of the National Academy of Sciences*, vol. 103 no. 23, June 2006.

[104] NEWMAN, M. E. J. “The Structure and Function of Complex Networks” *Society for Industrial and Applied Mathematics*, vol. 45 no. 2, 2003, pp. 167-256.

[105] NEWMAN, M. E. J., STROGATZ, S. H. and WATTS, D. J. “Random graphs with arbitrary degree distributions and their applications” *Physics Review* E, vol. 64 no. 2 026118, 2001.

[106] NEWMAN, M. E. J. and WATTS, D. J. “Renormalization group analysis of the small-world network model” *Physics Letters A*, Volume 263, December 1999, Pages 341-346.

[107] ÖGREN, P., FIORELLI, E. and LEONARD, N. E. “Cooperative Control of Mobile Sensor Networks: Adaptive Gradient Climbing in a Distributed Environment” *IEEE Transactions on Automatic Control*, vol. 49 no. 8, Aug. 2004, pp. 1292-1302.

[108] OLFAT-SABER, R., FAX, A. and MURRARY, R. “Consensus and Cooperation in Networked Multi-Agent Systems” *Proceedings of the IEEE*, 2007.

[109] PARDALOS, P. M. and PITSOULIS, L. S. “Nonlinear Assignment Problems” *Combinatorial Optimization*, vol. 7, Kluwer Publishers, Boston MA, 2000, pp. 1-34.

[110] PASCOAL, M., CAPTIVO, M. E. and CLIMACO, J. “A note on a new variant of Murty’s ranking assignments algorithm” *4OR: A Quarterly Journal of Operations Research*, vol. 1 no. 3, 2003, pp. 243-255.

[111] PIERSKALLA, W. “The multidimensional assignment problem” *Operations Research*, vol. 16, 1968, pp. 422-431.

[112] PRICE, I. C. and LAMONT, G. B. “GA Directed Self-Organized Search and Attack UAV Swarms” *Proceedings of the IEEE Winter Simulation Conference*, 2006, pp. 1307-1315.

[113] PUNNEN, A. P. and ANEJA, Y. P. “Categorized assignment scheduling: A tabu search approach” *Journal of the Operations Research Society*, vol. 44, 1993, pp. 376-379.

[114] RAFFAELLO, D. and BABISH, M. “The RoboFlag Testbed”. *Proceedings of the American Control Conference*, 2003, pp. 656-660.

[115] RAFFAELLO, D. and MURRAY, R. M. “The RoboFlag Competition”. *American Control Conference*. 2003.

[116] RAFIEI, D and CURIAL, S. “Effectively Visualizing Large Networks Through Sampling” *IEEE Visualization Conference*, 2005.

[117] REN, W. and BEARD, R. W. “Distributed Consensus in Multi-vehicle Cooperative Control” *Communications and Control Engineering*, Springer-Verlag London, 2008, pp. 1-118.

[118] REN, W., BEARD, R. W. and ATKINS E. “Information Consensus in Multivehicle Cooperative Control” *IEEE Control Systems Magazine*, 2007, pp. 71-82.

[119] RICCABONI, M. and SCHIAVO, S. “Structure and growth of weighted networks” *New Journal of Physics*, vol. 12, 2010.

[120] ROBINETT, R. D. III and HURTADO, J. E. “Stability and Control of Collective Systems” *Journal of Intelligent and Robotic Systems*, vol. 39, 2004, pp. 43-55.

[121] ROBINETT, R. D. III and WILSON, D. G. “Collective plume tracing: A minimal information approach to collective control” *International Journal of Robust and Nonlinear Control*, vol. 20, 2010, pp. 253-268.

[122] RYAN, A., TISDALE, J., GODWIN, M., COTTA, D., NGUYEN, D., SPRY, S., SENGUPTA, R. and HEDRICK, J. K. “Decentralized Control of Unmanned Aerial Vehicle Collaboration Sensing Missions” *Proceedings of the American Control Conference*, 2007, pp. 4672-4677.

[123] SANDELL, N. R., VARAIYA, P., ATHANS, M. and SAFONOV, M. G. “Survey of Decentralized Control Methods for Large Scale Systems” *IEEE Transactions on Automatic Control*, vol. ac-23 no. 2, April 1972, pp. 108-128.

[124] SCHNEIDERMAN, B. and ARIS, A. “Network Visualization by Semantic Substrates” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12 no. 5, 2006.

[125] SHABAB, M. “Cooperative Control of Multi-Vhehicle Systems”. Taken from http://staff.kfupm.edu.sa/SE/mshahab/072_EE550_CoopControl_SHAHAB_May2008.pdf, October 5, 2011.

[126] SHAW, L. B. and SCHWARTZ, I. B. “Enhanced vaccine control of epidemics in adaptive networks” *Physical Review E*, vol. 81, 2010.

[127] ŠILJAK, D. D. “Dynamic Graphs” *Nonlinear Analysis: Hybrid Systems*, vol. 2, 2008, pp. 544-567.

[128] ŠILJAK, D. D. and ZEČEVIĆ, A. I. “A new approach to control design with overlapping information structure constraints” *Automatica*, vol. 41, 2005, pp. 265-272.

[129] ŠILJAK, D. D. and ZEČEVIĆ, A. I. “Control of large-scale systems: Beyond decentralized feedback” *Annual Reviews in Control*, vol. 29, 2005, pp. 169-179.

[130] ŠILJAK, D. D. “Decentralized Control of Complex Systems” *Mathematics in Science and Engineering*, vol. 184, Academic Press, New York, 1991.

[131] SKORIN-KAPOV, J. “Tabu search applied to the quadratic assignment problem” *Operations Research Society of America Journal on Computing*, vol. 2, 1990, pp. 33-45.

[132] SHAFER, G. “A Mathematical Theory of Evidence” Princeton University Press, Princeton NJ, 1976.

[133] STEINBERG, L. “The backboard wiring problem: A placement algorithm” *Society of Industrial and Applied Mathematics Review*, vol. 3, 1961, pp. 37-50.

[134] STROGATZ, S. H. “Exploring complex networks” *Nature*, vol. 410, Mar. 2001, pp. 268-276.

[135] TAILLARD, E. “Robust taboo search for the quadratic assignment problem” *Parallel Computing*, vol. 17, 1991, pp. 443-455.

[136] TRAVERS, J. and MILGRAM, S. “An Experimental Study of the Small World Problem” *Sociometry*, vol. 32 no. 4, 1969, pp. 425-443.

[137] UGI, I., BAUER, J., FRIEDRICH, J., GASTEIGER, J., JOCHUM, C., and SCHUBERT, W. “Neue Anwendungsgebiete für Computer in der Chemie” *Angewandte Chemie*, vol. 91, 1979, pp. 99-11.

[138] VIEGAS, F. and DONATH, J. “Social Network Visualization: Can We Go Beyond the Graph?” *Workshop on Social Networks*, 2004.

[139] VLACH, M. “Branch and bound methods for the three-index assignment problem” *Ekonomicko-Matematicky Obzor*, vol. 3, 1967, pp. 181-191.

[140] VOLGENANT, A. T. “Solving the k-cardinality assignment problem by transformation” *European Journal of Operations Research*, vol. 157, 2004, pp. 322-331.

[141] WATTS, D. J. and STROGATZ, S. H. “Collective dynamics of ‘small-world’ networks” *Nature*, vol. 393, June 1998.

[142] XIE, G. and WANG L. “Consensus Control for a class of Networks of Dynamic Agents: Fixed Topology” *Proceedings of the IEEE Conference on Decision and Control*, vol. 44, 2005, pp. 96-101.

[143] XIE, G. and WANG L. “Consensus control for a class of networks of dynamic agents” *International Journal of Robust and Nonlinear Control*, vol. 17, 2007, pp. 941-959.

[144] ZAVLANOS, M. M. and PAPPAS, G. J. “Distributed Connectivity Control of Mobile Networks” *IEEE Transactions on Robotics*, vol. 24 no. 6, Dec. 2008.

[145] ZEČEVIĆ, A. I. and ŠILJAK, D. D. “Control design with arbitrary information structure constraints” *Automatica*, vol. 44, 2008, pp. 2642-2645.

[146] ZEČEVIĆ, A. I. and ŠILJAK, D. D. “Dynamic graphs and continuous Boolean networks, I: A hybrid model for gene regulation” *Nonlinear Analysis: Hybrid Systems*, vol. 4, 2010, pp. 142-153.