Summer 2010

The Influence of Network Factors on Network Centric Operations

Mehmet Fidanci
Old Dominion University

THE INFLUENCE OF NETWORK FACTORS

ON NETWORK CENTRIC OPERATIONS

by

Mehmet Fidanci
B.S. August 1993, Turkish Air Force Academy, Istanbul,Turkey
M.S. March 2000, Air Force Institute of Technology, Dayton, Ohio

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY

August 2010

Approved by:

Shannon Bowling (Director)

Resit Unal (Member)

Ghaith Rabadi (Member)

Sean Deller (Member)

Kerem Aytulun (Member)

ABSTRACT

THE INFLUENCE OF NETWORK FACTORS
ON NETWORK CENTRIC OPERATIONS

Mehmet Fidanci
Old Dominion University, 2010
Director: Dr. Shannon Bowling

As Information Age changes the lifestyle of all humankinds, it also changes the way how to defense and secure the borders are secured and defended. The Informartion Age is about information superiority. It evolves the command and control concept, proactively, to optimize the size of the units and their connections within a combat force for effective mission accomplishment. The biggest issue is how big a unit will be and how they will arrange and connect it to the command and control structure in order for the unit to be effective on the battlefield. While some arrangements connect to each other so well that they endure and perform effectively during combat, other arrangements that connect each other are so cumbersome that they either barely succeed or are killed.

Network Centric Operations concentrate on how to provide a warfighting unit with enough assets so that it can accomplish the assigned mission by itself effectively within its chain of command. The first thing that Network Centric Operations tries to achieve is to gain the shared awareness of the battlefield. This can be done by scouts, ground or air patrol, satellite image, radio frequency, etc. The situational awareness and the information superiority of the battlefield will definitely effect the enemy's operations so that the enemy needs to change its strategy. The second thing that Network Centric Operations tries to achieve is to have an impact on every occasion being reported or unexpectedly sensed in order to disrupt the enemy's will. How can a force achieve this? A well organized and a well connected force can have the information superiority and be able to transform that superiority to a success. For effectiveness, each asset in a combat force should have reliable connection capacity with command and control centers and other assets.

The number of Sensors and Influencers being the driving entities of the war unit in the battlefield are integer-partitioned and connected to a Decider. There are well defined rules, regulations, and well established connections between the entities. They are initially placed random to the simulation environment as the BLUE and RED forces. Each force starts sensing, tracking, reporting, and killing the opposing side. Each force tries to win the other side. Each combination of an experiment replicates 30 times and then results are reported. The probability of a BLUE force win was studied to measure the performance of a networked force.

The objectives of this research are to explore how units vary in size of organization, how they behave in a networked environment and to investigate how to increase the performance of a networked force. This research explores sufficient search space to understand the influence of network factors on Network Centric Operations.

ACKNOWLEDGEMENTS

A simple "thank you" cannot express how grateful I am to my advisor, Dr. Shannon Bowling. This research study would not have been completed without his experience, skill, and guidance. His knowledge of the topics, recommendations for appropriate and reachable goals, decisiveness, and clear communication style have made the difference to the undertaking reaching fruition and a final conclusion.

Sincere gratitude also goes out to the other members of the dissertation committee, Dr.Ghaith Rabadi, Dr. Resit Unal, and Dr. Sean Deller, for their advice and support in my research.

I would like to thank all the faculty in Aeronautics and Space Technologies Institute (ASTIN) and Old Dominion University (ODU) who had given me a fascinating learning experience in graduate school.

I am also very thankful to Turkish Air Force Headquarters for supporting and sponsoring my research study.

I would like to express my gratitude to Dr.Oktay Baysal and Dr. Osman Akan for their decisiveness in construction of collaborative Ph.D. program between ASTIN and ODU.

I also would like to thank my seniors and friend Dr. Murat Ermis, Dr. Kerem Aytulun, and Selami Yildiz for their support and guidance.

I thank my friends in graduate school at ODU: Ersin Ancel, Anil Ustun, Christopher Garcia, Elkin Rodrigues, and Yaw Mensah. Their friendship had made this part of my life very colorful and enjoyable.

I am always thankful and forever indebted to my dear wife and my only lovely sweet hearth daughter, Gulser and Maide Duru, for making me smile even during the most difficult times. Their love, patience, encouragement, and understanding has given me the greatest strength. Without my beloved family, this work would not have been possible.

I am forever indebted to my mother and father, Ayse and Ahmet, for their unconditional and selfless love.

I am very grateful to my in-laws, Remziye and Leyla, for their enthusiasm, and support during this journey.

Thank you all.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. INTRODUCTION

War is an inevitable reality of life and has been as long as humanity has existed. Countries go to war to defend themselves, or they use war to support their policies and beliefs. The tools and tactics of how we fight have always changed along as technology enables us to advance as the years go by.

War in the Information Age has different characteristics than the war in the Industrial Age. Technology was the dominant factor of the power in the industrial age. The Information Age focuses on the value and superiority of information (Lalbakhsh et al., 2009). These characteristics affect warfare capability, processes, and evaluation that are brought to combat as well as the nature of the environment in which conflicts occur.

Experience learned from past wars shows that traditional warfare is far from satisfying its initial intended purpose in the Information Age. The consequences of the information age and cultural changes from technology to information and the new concept of power to the edge affected and changed our lifestyle as well as the way we fight and defend.

The mains concept that causes a military organization to achieve the optimum combat success and efficiency by means of network technology has emerged over the last decade. This revolutionary concept is called Network Centric Warfare (NCW) or it`s civilian version Network Centric Operations (NCOs). A primary goal of this new transformation is to put a military organization at the leading edge of warfare technology, tactics, and awareness about the enemy. Its definition and applications are continually evolving.

Both success and failure of operations, in the Information Age, often rely heavily on necessary and sufficient data and information gathering, processing, and sharing.

Often in the past, countries' large military budgets allowed military organizations to pioneer both the development of technology and its applications.

Nowadays, commercial sectors seem to have taken over this role as pioneers in the technology. They have applied information technology effectively to run business worldwide.

In today's business, dominant enterpreneurs want to gain information superiority and transform it into a competitive advantage by adapting their traditional management and operations concept into NCOs. They have dramatically exploited information technology and coevolved their organizations and processes to best serve their customers (Honabarger, 2006).

Information Age technology has significantly reconfigured our concept of time and distance. Large amounts of information, data, and images can be securely shared online over a long distance. Time and distance are no longer a hindrance for communication. A boss can watch his or her employees during a manufacturing process and give them directives over a screen. A commander, as a decision maker, can be aware of warfighters' orientation in the battlespace over a computer and can develop a new tactics to increase the mission effectiveness and efficiency.

The concept of NCW has changed force composition and individual platform capabilities with force spatial distribution and tactics as important and scenario-dependent factors. NCW concentrates on the information-based aspects of force tactics: information collection, communication, and exploitation. The ability of a force to manage and exploit the information as centric depends on its connectivity: the existence, capacity, reliability of the links that connect its platforms, command and control centers, and other entities.

No matter what physical proximity or strict hierarchy during the unpredictable war environment, commanders can now use robust communication networks to scatter their forces and synchronize their behavior for synergy in real time, generating massed effect. These two factors, distributed forces and networked control, look to revolutionize all aspects of warfare. A suitable

analytical model is needed to describe distributed, networked combat (Cares, 2005).

1.1. BACKGROUND

Information is the most vital (crucial) asset of an organization in the information age. How it is attained and exploited affects the ability of any organization to cope with the competitive challenges it encounters. Improvements in communication and information technology in the 1990s made it easier and cheaper to distribute information wider than ever before. But this wider information distribution might have adverse unintended consequences.

Command, Control, Communications, Computers and Intelligence (C4I) for the Warrior, a concept that advocated vastly increased access to information at all echelons, prior to the articulation of Network Centric Warfare was the highest concern of the military authority: How is automated information flow controlled? Alberts (1996) wrote a book about the unintended consequences of information age technologies to clarify these concerns and made appropriate recommendations.

Mission Capacity Packages (MCPs) was recommended as a major conclusion from the analysis to answer these concerns. MCPs describe the answers of how to: operate, organize, command and control, design systems, as well as provide training and education. MCPs must coevolve according to changes in the force. Command and control should not be considered as a solved issue, but is needed to be coevolve as force capabilities and concepts of the operation change.

There are a lot of choices, of course, in how to shape and arrange an organization; this will have different impacts on the operation effectiveness of the organization. Some arrangements will improve self-synchronization, while other arrangements will exacerbate it. The goal of this study is to find the optimum arrangement according to the intent. How should an Information Age combat force be arranged in order to get its optimum effectiveness?

1.2. PROBLEM DEFINITION

"There is still, however, a gulf between a philosophical understanding of adaptation and the engineering prowess to make purposeful, stable and controllable adaptation a reality in the battlespace"(Cares, 2005). The main reason for this gulf is not having an acceptable and reasonable combat model in the Information Age. An Information Age Combat Model will be a good tool to help in observing and understanding a new system design, invention, and testing.

An Information Age Combat Model explicitly represents interdependecies in between agents and appropriately comes up with delicate tactical arrangements. The model will help to set a rule of thumb to guide the Information Age concept overview through development, systems engineering, operational experimentation, and program analysis.

The purpose of this research is to understand what causes Network Centric Operations to be effective and to understand the influence of network factors on NCOs. In this research, a second attempt will be studied to identify up to what configuration the utility of the Perron-Frobenius Eigenvalue (λ_{PFE}) is valued as a good metric to predict the perfomance of a network in general and particularly combat power of the Information Age (Cares, 2005). As the number of distinct λ_{PFE} values increases gradually, the ratio of the distinct λ_{PFE} values with respect to the different meaningful combinations decreases dramatically. Therefore, the power of the λ_{PFE} value as performance measure (predictor) will be expected to diminish exponentially from smaller networks to larger networks and be asymptotic to the horizontal line. The third attempt is to find some functions and algebraic operations to explain the relation in between the IACM configuration and its performance. These functions and operations can generate some numbers varry in a range as in the λ_{PFE} and those numbers, with or without λ_{PFE}, might give better explanation of its performance.

Since an Information Age Combat Model must explicitly point out networks, a mathematical structure of networks and its structure should be clearly defined. An ubiquitous term used for connected system is called as "network". It has other synonyms in business language such as "grid", "chain", or "mesh". But only very few can understand that the terms have very specific definitions in mathematical Network Theory. There are two practical reasons in selecting a network type: different networks have different properties, many of the characteristics of new operational concepts have specific mathematical definitions derived from the science of the networks. Any model of distributed networked combat that discards these mathematical properties would be inacceptable model of Information Age combat.

There are three main perspectives of networks comprehensibly. These are network structure, network dynamics, and network evolution.

2. LITERATURE REVIEW

As David et al. (2002) indicated, the information age has brought outstanding changes to the US military organization and operations. The term related with this change is Network Centric Warfare. From an information point of view, NCW is described as an information superiority-enabled concept of operations that creates advanced combat power by networking Sensors, decision-makers, and Influencers to accomplish mutual awareness, advanced speed of command, higher speed of operations, greater lethality, advanced survivability, and a degree of self synchronization. In essence, NCW capitalizes information superiority into combat power enhancement by effectively linking knowledgeable entities in the battlespace.

Hanratty et al. (2003) discussed the disadvantage of network centric warfare if not carefully arranged. Tomorrow's digitally networked battlefield will not only enable unprecedented access to data, information, and knowledge, but if not carefully arranged threatens to overload commanders and staff with this new technology and information overload. Structured and semi-structured data sources from all over the battlefield need to be monitored, filtered, and secured against information requirements with the given appropriate alert level to commanders and staff.

Wong-Juri et al. (2006) introduced a multi-layered model (MLM) with an interlayer mapping to address the interdependent contributions of processes, people, and systems to the success of Network Centric Operations. They proposed a methodology to model and analyze improvement in the development and implementation of Network Centric Warfare that extends the metrics described in the NCO Conceptual Framework. This methodology allows a commander to have the ability to determine and trace how desired military objectives are affected by changes in specific areas across the doctrine, organization, training, material, leadership, education, personnel, and facilities trade space. This type of information helps a commander develop a strategy in decision making.

Honda et al. (2006) evaluated agent-based combat simulation by introducing a synthetic approach and adaptive evolutionary learning to action rules by using EINSTein. EINSTein was developed by the Japanese Center for Naval Analyses. It is a multi-agent artificial war simulation consisting of a 2-dimensional lattice-shaped battlefield and agents of two groups, which are called the red force and the blue force, fighting in the battlefield. Action rules are expressed by a combination of parameters in combat simulation. The researchers iteratively changed the number of sets of action rules to decide how many of them work well. They made statistical analysis between homogeneity and diversity and showed that there is a trade-off between them. By using the synthetic approach, the total gain of a group is maximum at the stage that homogeneity and diversity are in the middle.

Qing et al. (2009) studied the C4ISR system effectiveness under the model of Network Centric Warfare and Platform Centric Warfare by utilizing graph theory, information entropy, knowledge function theory, and complexity theory. They concluded that information sharing has an active (positive) impact and network complexity has a negative impact which are both raised as a whole when the degree increases.

McCormick et al. (2004) introduced a new service-oreiented architecture (SOA) approach that has gained popularity in the commercial sector by integrating totally different enterprise applications, and representing a practicable approach to network-centric warfare applications. They described how agents provide a critical technology to apply emerging commercial technologies, such as web services, into network centric warfare problems. Their objecive is to develop and share battlespace awareness and understanding. Their information service supervises information collection and dissemination/publishing activities on behalf of fusion services in an autonomous, yet controllable fashion.

2.1. DEFINITION OF NETWORK CENTRIC WARFARE

No matter whether it is called Network Centric Warfare, Network Centric Operations, or Netcentric Warfare, it is a new military concept of war pioneered by the United States Department of Defense.

It attempts to transform an information advantage, gained by information technology, into a challenging warfighting advantage by the virtue of robust secure networking and geographically dispersed forces. This new design networks with updates in technology, organization, process, and people and can create a better organizational behavior.

There are three tenets in Network Centric Warfare to create synergy that dramatically increase mission effectiveness. These tenets cause and enable chain reactions to each other. Network Centric Warfare is built and depends on a well designed, easy to access, wide band, robust network. Geographically dispersed forces share information, collaborate with their echelons to have better information, and orient themselves to the battlespace for situational awareness. Shared situational awareness enables self synchronization. Overall, everything dramatically increase mission efectiveness.

Network Centric Warfare has some architectural and design challenges. Providing secure communications in Network Centric Warfare is a challenging task. First of all, coordinating bandwith usage in a battlespace is a difficult issue. Whenever a unit logs in and data transfer starts, it will be source or relay of radio frequency (RF). For example, there were more than 500,000 troops who were supported with 100 Mbit/s of bandwith during the Desert Storm Operation. Today, there are about 350,000 warfighters, supported by more than 3,000 Mbit/s of satellite bandwith in the Iraqi Freedom Operation. The bandwith, number of access and speed of network, is 30 times more than they had about a decade ago. They essentially used the same weaponries in timely close operations with significantly increased effectiveness.

Second, providing secure and reliable information transfer in network centric warfare is another difficult issue. Succesful key management for encryption must be supported for secure information over the network.

Third, every unit in network has different levels of access authority for information. This makes difficult to efficiently transfer information between networks with different levels of security classification. There are a lot of issues still needed to be determined for secure and reliable network. Although multi level access security systems seem to resolve the issue, to what extent specific data should or should not be transfered still needs to be determined during the decision making process.

Fourth, situational awareness is limited when maneuvering in weak or non-existent GPS coverage. Spare systems in case of GPS outage for a variety of reasons needs to be considered as a backup for reliable fusion of positional data (triangulation technics can be used to locate yourself from multiple sensors as backup). [1]

2.2. NETWORK STRUCTURE

The most fundamental level of the Information Age Combat Model is the *mathematical structure of a network as a collection of nodes connected by links.* Nodes are the processing elements called Sensors, Deciders, Influencers, or Targets. These nodes are well defined (Cares, 2005) and have the following properties:

- *Sensors* detect unusual or hostile activities in their responsility areas and locate them or receive those activities' locations from friendly nodes and send the information to their linked *Deciders,*
- *Deciders* receive information from their linked *Sensors* and make decisions and command their linked Influencers about the present and future arrangement,

[1] See http://en.wikipedia.org/wiki/Network-centric warfare for more information

- *Influencers* receive direction from their linked D*eciders* to render the given hostile nodes states useless,
- *Targets* are nodes that have military value but are not *Sensors*, *Deciders,* or *Influencers*.

These are the minimum properties required to define each node. There are still some characteristics needed to be defined to clarify the rules between nodes.

First, each node must belong to a "side" of at least two (e.g., blue, red, friend, foe, neutral). For simplicity and a better fit to the combat model, there are two sides, conventionally termed BLUE (depicted in black) and RED (depicted in gray).

Second, Targets always belong to the other side, adversary. Targets are anything of military value on each side except a Sensor, Decider, or Influencer.

Third, sensor logic (signal reception) is not a decision making capacity. Signal reception is already considered as an embeded function within Sensors.

Fourth, all Sensor information must pass through a Decider. Deciders know their side's nodes location even if they are killed or inoperative accepting they are all in their own side Sensors' coverage.

Nodes are connected to each other by directional links. Links might be observable phenomenon like radio frequency energy, infrared signals, light signals, communications or acoustic energy that emanate from a node and are detected by a Sensor. These detected links by Sensors are sent to Deciders. Deciders issue orders to Influencers, Sensors, and Targets. Influencers typically destroy or render useless the nodes they interact with. Most of the links in the Information Age Combat Model are tactical and operational interactions between nodes.

2.3. COMBAT NETWORKS

The links and nodes described above establish a combat network. Figure 2.1 graphically represents the most basic one-sided combat network, while Figure 2.2 represents a two-sided system. Black nodes denotes the friendly side, while light grey denotes the enemy side. Different line styles represent various kinds of links between nodes.

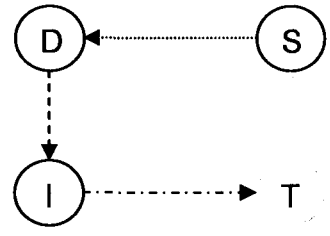


Figure 2.1. One-Sided Basic Combat Network (Cares, 2005)

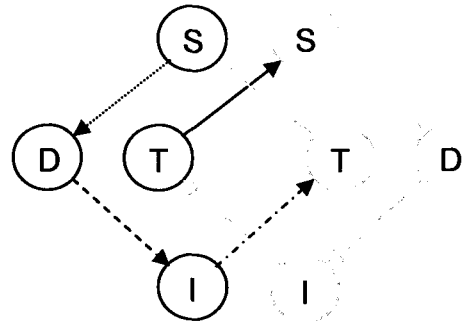


Figure 2.2. Two-Sided Basic Combat Network (Cares, 2005)

Figure 2.3 represents the basic complete combat network that can be established from what has been mentioned so far. It represents all possible meaningful links in which Sensors, Deciders, Influencers, and Targets interact with each other.

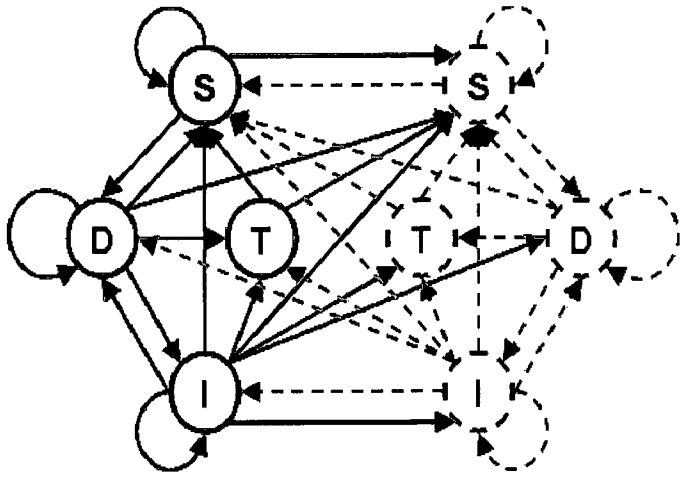


Figure 2.3. Two-Sided Basic Complete Combat Network (Cares, 2005)

2.4. DIMENSIONS AND COMPLEXITY

The number of possible links for eight nodes is equal to 2^8=64. As Cares (2005) described, two-sided basic complete combat network for eight nodes (SDIT nodes for the BLUE side and SDIT nodes for the RED side) (see Figure 2.3). This is depicted in the adjacency matrix (Figure 2.4) as having at least 36 different dimensions (i.e.,possible meaningful links). An adjacency matrix is an easier representation for understanding the dimensionality of different types of network. Figure 2.4 reflects the same eight node network in Figure 2.3 in matrix form.

$$a_{ij}=\begin{cases}1, & \text{if there is a link from row i to column j}\\ 0, & \text{otherwise}\end{cases}$$

	S	D	I	T	S	D	I	T
S	1	1	0	0	1	0	0	0
D	1	1	1	1	1	0	0	0
I	1	1	1	1	1	1	1	1
T	1	0	0	0	1	0	0	0
S	1	0	0	0	1	1	0	0
D	1	0	0	0	1	1	1	1
I	1	1	1	1	1	1	1	1
T	1	0	0	0	1	0	0	0

Figure 2.4. Adjacency Matrix

The number of possible links for eight nodes is reduced from 64 to 36 based on the following important assumptions as Deller (2009) mentioned and tabulated in his research as follows:

- Targets are inactive nodes; they can only be sensed. As seen in Figure 2.3 and Figure 2.4, there are only two arrows out from Targets to Sensors. Therefore, ***12*** links from Targets to Deciders, Influencers, and Targets are excluded. There are no links from Targets to Deciders, Influencers, and Targets.
- Sensors are also inactive nodes; they just relay information to linked Deciders and to both sides of the Sensors. There are three arrows out from Sensors to linked Deciders and both sides of the Sensors. Therefore, there are ***10*** links from Sensors to Influencers, and Targets are excluded.
- Deciders act through all linked nodes and can sense adversary Sensors. There are five arrows out from Deciders to all linked nodes and adversary Sensors. Therefore, there are ***6*** links from Deciders to adversary Deciders, Influencers, and Targets are excluded.

Deller further reduced the number of link types from 36 to 18 based on the BLUE/RED symmetry. Links from a node to itself in Figure 2.3 have been interpreted as connecting two different nodes of the same type and side.

Table 2.1. Available Links Types in the IACM (Tabulated by Deller 2009)

Link Type	From	To	Interpretation	Link Type	From	To	Interpretation
1	S_{BLUE} S_{RED}	S_{BLUE} S_{RED}	S detecting own S, or S coordinating with own S	10	I_{BLUE} I_{RED}	D_{BLUE} D_{RED}	I attacking own D, or I reporting to own D
2	S_{BLUE} S_{RED}	D_{BLUE} D_{RED}	S reporting to own D	11	I_{BLUE} I_{RED}	I_{BLUE} I_{RED}	I attacking own I, or I coordinating with own I
3	S_{BLUE} S_{RED}	S_{RED} S_{BLUE}	S detecting adversary S	12	I_{BLUE} I_{RED}	T_{BLUE} T_{RED}	I attacking own T
4	D_{BLUE} D_{RED}	S_{BLUE} S_{RED}	S detecting own D, or D commanding own S	13	I_{BLUE} I_{RED}	S_{RED} S_{BLUE}	I attacking adversary S, or S detecting adversary I
5	D_{BLUE} D_{RED}	D_{BLUE} D_{RED}	D commanding own D	14	I_{BLUE} I_{RED}	D_{RED} D_{BLUE}	I attacking adversary D
6	D_{BLUE} D_{RED}	I_{BLUE} I_{RED}	D commanding own I	15	I_{BLUE} I_{RED}	I_{RED} I_{BLUE}	I attacking adversary I
7	D_{BLUE} D_{RED}	T_{BLUE} T_{RED}	D commanding own T	16	I_{BLUE} I_{RED}	T_{RED} T_{BLUE}	I attacking adversary T
8	D_{BLUE} D_{RED}	S_{RED} S_{BLUE}	S detecting adversary D	17	T_{BLUE} T_{RED}	S_{BLUE} S_{RED}	S detecting own T
9	I_{BLUE} I_{RED}	S_{BLUE} S_{RED}	I attacking own S, or S detecting own I	18	T_{BLUE} T_{RED}	S_{RED} S_{BLUE}	S detecting adversary T

Cares (2005) employs only basic combat networks similar to Figure 2.1 with one replacement. He replaced Target by an adversary Sensor or Influencer. His combat cycles contain only links of types 2,3,6,13, and 15. Type 13 has two interpretations. Its both interpretations will be used and distinguished by the model context.

2.5. NETWORK DYNAMICS

Advantages of networked centric warfare occur in local tactical operations because of the persistent dynamic interaction between specifically arranged nodes over links. This dynamic interaction process is called a cycle, sub-network in which the functions of nodes are sent to each other over a path that revisits at least one node once. Useful networked functions depend on presence of a cycle.

3. METHODOLOGY AND PROPOSED APPROACH

In this chapter, the methodology that will be used in the dissertation will be explained.

3.1. WHY DO WE USE (AGENT-BASED) MODELING?

Models are designed, developed, and implemented as simulations to evaluate and gain insight about systems' behaviors in regulated environments. Modeling is a simple collation of the important entities, processes, and their relations to aspects of the real world. As Tolk et al. (2008) mentioned in their paper, current modeling paradigm is mostly intention-based. Entity capability and process are, in most cases, shaped by the models according to the intention and desired effect, which is in turn essentially reduces the probability of success to desired effect. They proposed a new modeling paradigm based on the agent metaphor: effect-based modeling. The new modeling paradigm uses agents as having multi-roles entities, as well as processes, with their potential effects. In other words; everything is defined as an agent with more flexible evaluation algorithm to capture the effects and higher-order effects of complex and non-linear systems that generate.

The modeler has a preset purpose in mind while building a model. He or she wants to see if that purpose is achievable. He or she wants to evaluate several alternatives, optimize his decision based on several situations, train people using a simulator, etc. In any case, he or she is first inspired a model conceptually by the real world. The concept can either be a feature that is situation independent and describes entities, or a fluent that is situation dependent and describes processes. In other words, modeling involves entity, process, and their relations. An entity might have many roles; but, it is often reduced to a main intended role in the modeling process. A process is a course of action to change the current situation into a desired direction for the desired outcome.

The current, intention-based, modeling paradigm has three main shortcomings. These are intention-based capability modeling, intention-based process modeling and intention-based evaluation.

Intention-based capability modeling, in general, concentrates on the main role or the intended use and not inherent capabilities, which can restrict its applicability for new domains with changing scopes.

Intention-based process modeling, in general, models the probable desired outcome. It normally ignores unintended outcomes, side effects, and follow-on effects.

Intention-based evaluation modeling often narrows down its performance metrics after action reviews for efficiency evaluation to measure intended effects. Therefore, evaluation procedures are too strict regarding new scopes.

On the other hand, Tolk et al. (2008) proposed a new modeling paradigm "effect-based modeling" to compensate for the shortcomings of the current modeling paradigm. Effect-based modeling in the military domain means effect-based operations that Smith (2002) defined as *"coordinated set of actions directed at shaping the behavior of friends, neutrals, and foes in peace, crisis, and war."*

Effect-based operations introduce the idea of multi-level, cascading effects as shown in Figure 3.1 below. Not only entities can produce effects, but effects themselves can produce essentially decreasing effects.

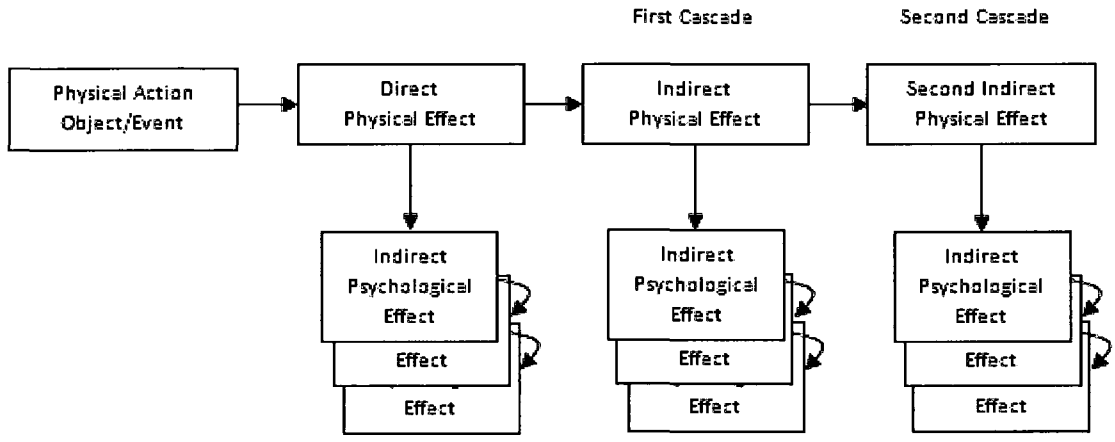


Figure 3.1. Chain Effects (Smith 2002)

Tolk et al. (2008) recommends that modeled entities should not be used just for intended purposes, but they should be able to conduct all possible purposes and functions with identified capabilities based on their available properties. Using entities with ready to use or that have a multi-purpose use is a more complex but a more efficient way for simulation. New modeling paradigms aim for each simulated entity to be equipped with actual capabilities with potential capabilites described in sufficient detail using properties and associations. Anything (role, capability, function, purpose, uses, etc.) needs to be described in each simulated entity and should be embedded to its property so that each entity is ready to support any potential roles described in its properties.

The whole process and its possible interactions with all entities, as well as other processes, are also necessary to model with the same detail as entities, their properties, and associations.

Intention-based evaluation criteria should also be changed accordingly to meet the requirements of effect-based evaluation criteria. Specifically, when agent-based simulation is used in human behavior modeling with computer support, running into structural variances based inadequate evaluation criteria is obvious, as shown in Tolk (1999). The internal decision logic, the external evaluation logic, model entities, and processes should be consistent with each other. The internal logic controls the entities behavior with respect to the situated

simulation environment. The external evaluation logic checks and evaluates if the objectives have been met. Therefore, corelation metrics are needed to be working as fitness function between internal decision logic and external evaluation logic.

In order to analyze effect-based net-centric operations, intention-based modeling falls short. Discrete event simulation is a high level and not sufficient enough to explore the micro aspect relation and the interactions between entities. Agent-directed simulation provides the metaphors needed to build the necessary models. Using agents to not only represent Influencers and Targets but also the processes, it becomes possible to capture all effects and move from "what I intended to accomplish" to "what I really accomplished" including side and secondary effects. Computational challenges exist, but they seem to be easier to overcome than the conceptual weaknesses of alternatives (Tolk et al. 2008).

3.2. AN AGENT-BASED SIMULATION MODEL USING THE IACM

The λ_{PFE} is a reasonable metric for the IACM structure with which to measure the performance of a networked force (Deller, 2009). To determine if it is an indicator of combat effectiveness, the agent-based simulation of the IACM coded in NetLogo was modified with a more powerful and more flexible one coded in AnyLogic to conduct a series of force engagements between opposing forces of equal assets and capabilities with differences in their connectivity arrangements or configurations for large cases.

The agent-based model was used for two purposes: the primary focus of this investigation is to explore how various sizes of units inside organizations behave in a networked environment. The secondary focus of this investigation was to determine how to increase the performance of a networked force.

As Deller (2009) mentioned in his research, both sides of equal forces seek for what is best for their benefit as opposed to what is worst for the enemy side. For this reason, it is necessary to calculate λ_{BLUE} and λ_{RED} separately to analyze the performance of both sides for all their configurations. In order to

separately calculate λ_{PFE} value for BLUE and RED sides, the single-sided adjacency matrix in Figure 3.2 is given below as an example for the 4-3-4-1 configuration used with a single Target node; its eigenvalue for combination {2,1,1,2,1,1} is 1.565. Target node symbolizes all the enemy forces capable of being targeted.

	S	S	S	S	D	D	D	I	I	I	I	T
S	0	0	0	0	1	0	0	0	0	0	0	0
S	0	0	0	0	1	0	0	0	0	0	0	0
S	0	0	0	0	0	1	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	1	0	0	0
D	0	0	0	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	1
T	1	1	1	1	0	0	0	0	0	0	0	0

Figure 3.2. Single-Sided Adjacency Matrix for 4-3-4-1 Configuration

3.3. STRUCTURE OF THE EXPERIMENT

Any difference in force effectiveness can be best explained with the difference in connectivity. The more Sensors and Influencers are linked to a Decider, the better performance it will respond with. For unbiased simulation and simplicity, the same assumptions as in Deller (2009) are held as containing an equal number of Sensors and Influencers with both having the identical performance capabilities. So the structure of both sides is represented by an X-Y-X-1 template as S-D-I-T.

No matter what the structure will be and therefore the template of both sides, a better Java code was scripted to distinguish the different meaningful combinations and a more flexible agent-based simulation model was developed in a more powerful environment. The adjacency matrix will always have the same

number of rows as the number of columns and it is always a square no matter what their arrangements are. So, solely the value of λ_{PFE} can be calculated.

For example, a 6-4-6-1 friendly force and a 7-3-7-1 enemy force arrangements are given. Their meaningful combinations are also always independent from each other.

6-4-6-1 friendly force

	S	S	S	S	S	S	D	D	D	D	I	I	I	I	I	I	T
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
T	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

7-3-7-1 enemy force

	S	S	S	S	S	S	S	D	D	D	I	I	I	I	I	I	I
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Figure 3.3. Single-Sided Adjacency Matrices for 6-4-6-1 vs. 7-3-7-1 Configurations

There is a finite number of ways to link Xs and Ys to each other for their certain numbers. Deller (2009) made two important scoping decisions for the rules of the game, IACM; those decisions were also held in this study. First, each Sensor and Influencer would only be linked to one Decider (a ***vertical / execution / operation / hierarchial link in the chain of command***), not two or more Deciders (but the given Decider does not have that limitation; it could be linked to multiple Sensors and Influencers). Second, the connectivity within any X-Y-X-1 arrangements was subjected to only those hierarchial links in the chain of command (links in between dissimilar entities) necessary to create the combat (adjacency matrix) cycles (i.e., link types 2,3,6,13 and 15 in Table 2.1 as stated earlier), which are the fundamental links to calculate λ_{PFE} (Deller, 2009).

For example, for a 7-3-7-1 arrangement, there are 7 Sensors, 3 Deciders, 7 Influencers.

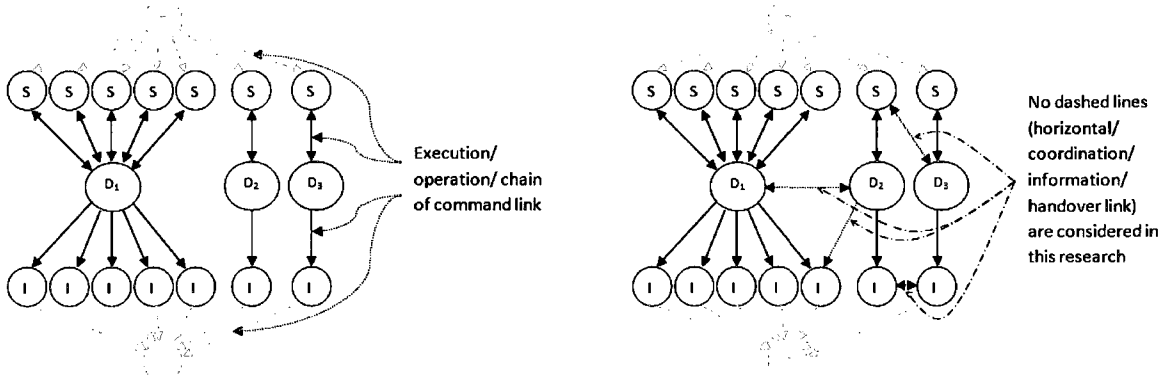


Figure 3.4. A Sample Type of Links

Future works should include "horizontal / coordination / information / handover / peer-to-peer" links in between similar entities like Sensors to Sensors, Deciders to Deciders, and Influencers to Influencers such link types 1, 5, 11 or direct coordination links from Sensors to Influencers such a link type 9. A new rule or function to determine what is going to happen to a Decider with enough influencers but no Sensors or vice versa can be another future study. These additional links and rules will definitely increase the performance of a networked force as well as its structure and eigenvalues.

The number of possible configurations for an X-Y-X-1 force becomes large very fast as X increases. The number of different meaningful combinations for any number of a template is a combinatorial coupling relation of X and Y. Three modular Java codes were written to determine the different meaningful combinations. For example, there are a total of thirty six possible ways to distribute five Sensors and five Influencers across three Deciders. When we integer partition and permute five by three, we get six possible configurations between five Sensors and three Deciders (or five Influencers and three Deciders); let's say a sub matrix, A, m by three in dimension. Since we have the same number of Influencers, we will get the same six possible configurations between three Deciders and five Influencers; the same sub matrix, A, m by three in dimension. Then the total number of possible configurations for a 5-3-5-1 force

will be six times six, equal to thirty six. In order to distinguish the different meaningful combinations from the possible configurations, we will pretend as if multiplying the sub matrix (as being the connectivity matrix of Sensors and Deciders) by its transpose (as being the connectivity matrix of Deciders and Influencers); but in reality we apply special matrix operation. This special matrix operation gives us thirty six real numbers with fractions; some are repeated, but some are distinct. Those numbers with fractions work as an index. The fractional numbers detect the difference among all possible combinations. As the number of Sensors/Influencers and the number of Deciders get closer to each other, the number of all possible meaningful combinations and therefore the number of different meaningful combinations decrease. The constituents of the distinct results (real numbers) are our different meaningful combinations. The special matrix operation is defined as below:

$$\mathbf{AoA'} = \begin{bmatrix} \mathbf{a_{11}} & \cdots & \mathbf{a_{1y}} \\ \vdots & \ddots & \vdots \\ \mathbf{a_{m1}} & \cdots & \mathbf{a_{my}} \end{bmatrix} \mathbf{o} \begin{bmatrix} \mathbf{a_{11}} & \cdots & \mathbf{a_{1m}} \\ \vdots & \ddots & \vdots \\ \mathbf{a_{y1}} & \cdots & \mathbf{a_{ym}} \end{bmatrix} = \sum_{\mathbf{j=1}}^{\mathbf{y}} \frac{\mathbf{a_{ij}^{y}}}{\mathbf{a_{ji}^{1/y}}} \qquad \textbf{Equation 3.1}$$

Where

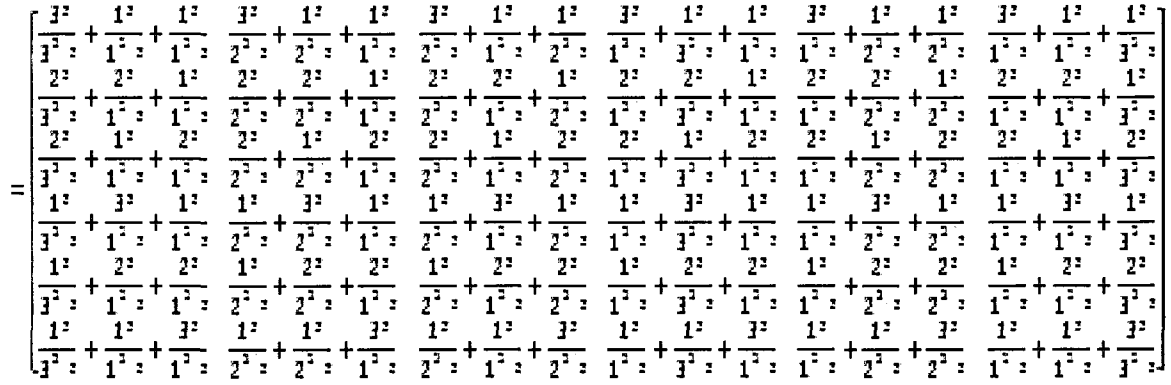
$$\mathrm{A} = \begin{bmatrix} a_{11} & \cdots & a_{1y} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{my} \end{bmatrix}$$

$$\mathrm{A'} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{y1} & \cdots & a_{ym} \end{bmatrix}$$

$1 \leq i \leq m$ and y as nDeciders

The 5-3-5 case is given below as an example to explain how to obtain different meaningful combinations. The case has six possible combinations in between Sensors and Deciders and therefore it has the same number of possible combinations between Deciders and Influencers. These combinations are depicted in matrix form for convenience.

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 3 \end{bmatrix} \circ \begin{bmatrix} 3 & 2 & 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 3 \end{bmatrix}$$



$$= \begin{bmatrix} 20.7208 & 23.2236 & 23.2236 & 28.6934 & 28.5874 & 28.6934 \\ 14.5469 & 13.6992 & 15.1433 & 14.5469 & 15.1433 & 16.6934 \\ 14.5469 & 15.1433 & 13.6992 & 16.6934 & 15.1433 & 14.5469 \\ 28.6934 & 23.2236 & 28.5874 & 20.7208 & 23.2236 & 28.6934 \\ 16.6934 & 15.1433 & 15.1433 & 14.5469 & 13.6992 & 14.5469 \\ 28.6934 & 28.5874 & 23.2236 & 28.6934 & 23.2236 & 20.7208 \end{bmatrix}$$

Figure 3.5. The Calculation of Different Meaningful Combinations for the 5-3-5 Case With Special Matrix Operation

If matrix multiplication is applied, it yields the below matrix.

$$= \begin{bmatrix} 11 & 9 & 9 & 7 & 7 & 7 \\ 9 & 9 & 8 & 9 & 8 & 7 \\ 9 & 8 & 9 & 7 & 8 & 9 \\ 7 & 9 & 7 & 11 & 9 & 7 \\ 7 & 8 & 8 & 9 & 9 & 9 \\ 7 & 7 & 9 & 7 & 9 & 11 \end{bmatrix}$$

Figure 3.6. The Calculation of Different Meaningful Combinations for the 5-3-5 Case With Matrix Multiplication

This matrix operation can not be just addition, subtraction, multiplication, or division or any combination of these. Because, the same number in different place or different numbers in the same place might give the same result. The two resulting matrices in Figure 3.5 and Figure 3.6 for the same case are clear the

rationale behind why it is necessary to have a special function or an operator. The first resulting matrix detected the exact result as eight; but the second one is so rough it missed half of the different meaningful combinations and detected four. These basic calculus operators are not sensitive enough to distinguish the different meaningful combinations. The desired operation can not be a logarithmic, natural logarithmic, or exponential function because these functions are not sensitive to one, for example, ln(1)=0, log(1)=0. A special function and an operation are required to detect the difference for the intended purpose. What is the intended purpose? It is to identify the different meaningful combinations. What is meant from different meaningful combinations is how many different ways of links in between Deciders, Sensors, and Influencers have. The sequence is not important. In the case above, there are eight different meaningful combinations out of 36 possible configurations. The numbers in the resultant matrix are nothing but the keys show us their constituents of different meaningful combinations. There are three "20.7208" in the resultant matrix showing that they have the same configuration hanging together that no matter where they are, one Decider has three Sensors and three Influencers linked to it. The other two Deciders have one Sensor and one Influencer. The order is not important; but the number of Sensors and Influencers linked to each Decider is the key structure here. They go out to the battle field; it is known that one of the war units has one Decider with three Sensors and three Influencers fighting together as a team, the other two Deciders have one Sensor and one Influencer fighting together as the other team.

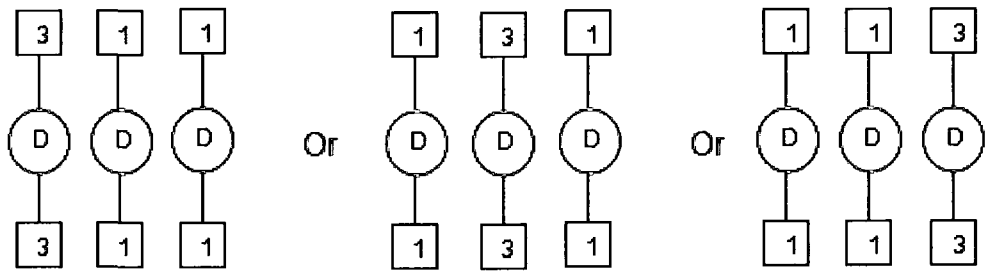


Figure 3.7. A Sample of Different Meaningful Combination for the 5-3-5 Case

The formula with the powers will be so sensitive to detect the different meaningful combinations as the numbers increase. But this formula takes time as the numbers increase to get the results. For a future work, some other mathematical formula or an algorithm for this purpose can be developed to get faster results. The remaining twenty eight possible configurations in the above 5-3-5-1 case are all modeled identically to these eight configurations in the IACM.

Adding a single Sensor and Influencer yields a 6-3-6-1 networked force, which can be organized in 100 possible ways. By applying the same formula, those 100 possible configurations are reduced to only 19 meaningful different configurations. The ratio between the number of meaningful different configurations and number of possible configurations diminishes as the number of Sensors and Influencers increases.

Identifying the different meaningful combinations is so crucial for the purpose of the problem. It is necessary to run different meaningful combinations to get all possible different results. It is not necessary to run the recursive combinations. They give nothing and waste time. For example, with a 6-3-6-1 arrangement, there are 100 possible combinations. Testing each of the 100 possible configurations of a 6-3-6-1 BLUE networked force against all 100 possible configurations of an opposing 6-3-6-1 RED networked force would require 10,000 similar engagements, but 19 different meaningful combinations would only require 361 unique engagements. The numbers of different meaningful combinations for all X-Y-X-1 forces where X<19 and Y <19 are calculated by using the Java coded algorithms based on the numbers of unique values for the distributions of Sensors and Influencers across the Deciders.

The resulting totals are consistent with Deller (2009) to where he left and are summarized in Table 3.1:

Table 3.1. The number of different meaningful combinations of all X-Y-X-1 networked forces where X<19 and Y<19

Numbers of Sensors (X) and Influencers (X)	Number of Deciders (Y)															
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	1															
4	2	1														
5	5	2	1													
6	13	5	2	1												
7	42	27	5	2	1											
8	75	74	30	5	2	1										
9	139	165	95	31	5	2	1									
10	224	363	245	105	31	5	2	1								
11	352	723	614	307	105	32	5	2	1							
12	517	1297	1367	914	325	110	31	5	2	1						
13	744	2247	2879	2356	964	340	116	34	5	2	1					
14	1032	3742	5874	6643	2552	1105	375	123	32	5	2	1				
15	1405	5967	11665	10022	6704	3009	1235	420	127	33	5	2	1			
16	1562	9247	19769	21155	16465	8354	3753	1447	441	134	34	10	2	1		
17	2432	13559	33205	42337	39035	22327	10216	4572	1543	456	136	37	10	2	1	
18	3115	20307	55515	81376	88795	57702	31556	13524	4997	1635	475	154	36	10	2	1

Each combination has its own adjacency matrix representation showing its node connectivity. The adjacency matrices for all configurations will only change in **SD** and **DI** sub-matrices (see the two white sections of an example adjacency matrix in Figure 3.8), with *S by D* and *D by I* in dimensions. These sub-matrices reflect the connectivity of each Sensor and Influencer to and from a particular Decider, and change by combination based on the allocation of Sensors and Influencers across the Deciders. The sub-matrices with zeros in gray areas represent the absolute absence of any links from the letters in the rows to the letters in the column. The sub-matrices with ones (1) in gray areas represent the existence of links from the letters in the rows to the letters in the column. No matter what X-Y-X-1 arrangements are, there are 16 sub-matrices in the adjacency matrix; 14 of them are steady as zeros or ones in varying dimensions depending on Sensors, Deciders, Influencers, and Target. Since two of sixteen sub-matrices of the adjacency matrices for each combination are varying, the variance between the λ_{PFE} values is small.

	S	S	S	S	S	S	D	D	D	D	I	I	I	I	I	I	T
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
T	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 3.8. An Adjacency Matrix for one of the 42 Different Meaningful combinations of a 7-3-7-1 network.

No matter what type of arrangements, there are always four eigenvalues. By definition, the maximum number of eigenvalues is n out of *n by n* square matrix. Four out of n have some values and the rest are zeros. The first four of the eigenvalues basically have the same pattern: two real and two complex numbers. The first eigenvalue is negative real number, the second one is positive complex number, the third one is complex conjugate of the second one, and the fourth one is positive of the first one.

In the case of a 7-3-7-1 networked force, 18 eigenvalues are given below for its first combination (5-1-1 vs. 5-1-1) as an example;

Table 3.2. The Eigenvalues of a 7-3-7-1 Networked Force for its First Combination

-2.2795	-0.0 + 2.2795i	-0.0 - 2.2795i	2.2795
-0.0 + 0.0i	-0.0 - 0.0i	0.0 + 0.0i	0.0 - 0.0i
0.0 + 0.0i	0.0 - 0.0i	-0.0	-0.0 + 0.0i
-0.0 - 0.0i	-0.0	-0.0	0.0
0	0		

The positive real eigenvalue is taken and is called as Perron–Frobenius eigenvalue (λ_{PFE}) as Deller mentioned in his study. In a 7-3-7-1 networked force case, the 42 different meaningful combinations have 13 unique λ_{PFE} ranging from

1.821 to 2.280. The λ_{PFE}'s were calculated by using a code in Matlab (available in the Appendix). The Matlab code reads X_Y_X.txt file (meaningful combinations file) for each arrangement and gives output as X_Y_X.xlsx woorkbook in real_eigenvalues, imag_eigenvalues, PFE_eigenvalues, variance worksheets.

As Deller (2009) mentioned in his research, identical combinations always have the same λ_{PFE} ; but, somehow different meaningful combinations also have the same λ_{PFE} . The combinations having the same eigenvalue are called the eigenspace. By definition "[t]he eigenspace corresponding to one eigenvalue of a given matrix is the set of all eigenvectors of the matrix with that eigenvalue."[2]. As the number of different meaningful combinations increases, the number of distinct eigenvalues decreases, and thus the ratio between the two. The λ_{PFE} loses its power gradually as a metric as the value of X increases. For a small number of cases, the eigenvalue alone can be a good metric; but, as the case and numbers increases, it needs to be supported by better defined (sensitive) metrics to enhance performance prediction of a networked force.

The numbers of unique λ_{PFE}'s for the different meaningful combinations for all X-Y-X-1 forces where X<19 and Y<19 are listed in Table 3.2.

[2] See http://en.wikipedia.org/wiki/Eigenvalue,_eigenvector_and_eigenspace for more information

Table 3.3. The Numbers of Unique λPFE's of all X-Y-X-1 Networked Forces where X<19 and Y<19.

Numbers of Sensors (X) and Influencers (X)	Number of Deciders (Y)															
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	1															
4	2	1														
5	4	2	1													
6	8	4	2	1												
7	13	8	4	2	1											
8	20	13	8	4	2	1										
9	27	20	13	8	4	2	1									
10	38	27	20	13	8	4	2	1								
11	48	38	27	20	13	8	4	2	1							
12	61	48	38	27	20	13	8	4	2	1						
13	76	61	48	38	27	20	13	8	4	2	1					
14	93	76	61	48	38	27	20	13	8	4	2	1				
15	***109***	93	76	61	48	38	27	20	13	8	4	2	1			
16	131	110	93	76	61	48	38	27	20	13	8	4	2	1		
17	***151***	131	110	93	76	61	48	38	27	20	13	8	4	2	1	
18	174	152	131	110	93	76	61	48	38	27	20	13	8	4	2	1

There is no simple relation between the numbers of unique λ_{PFE}'s and the numbers of different meaningful combinations (Deller, 2009). It is interesting that the numbers of different λ_{PFE}'s are recursive over diagonals with two exceptions. The numbers of unique λ_{PFE}'s are increasing by rows (each row it increases; it increases as Sensors/Influencers increase) and decreasing by columns (each column it decreases, it decreases as the Deciders increase). Table 3.3 depicts the percentages of unique λ_{PFE}'s over the numbers of the different meaningful combinations of all X-Y-X-1 networked forces where X<19 and Y<19:

Table 3.4. The Percentages of Unique λPFE's over the Numbers of the Different Meaningful Combinations fo all X-Y-X-1 Networked Forces where X<19 and Y<19.

Numbers of Sensors (X) and Influencers (X)	Number of Deciders (Y)															
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	100.00%															
4	100.00%	100.00%														
5	50.00%	100.00%	100.00%													
6	42.11%	44.44%	100.00%	100.00%												
7	30.95%	29.63%	44.44%	100.00%	100.00%											
8	25.64%	17.57%	25.67%	44.44%	100.00%	100.00%										
9	19.42%	11.90%	13.68%	25.81%	44.44%	100.00%	100.00%									
10	15.95%	7.44%	8.05%	12.38%	25.81%	44.44%	100.00%	100.00%								
11	13.71%	5.41%	4.40%	6.64%	12.04%	25.00%	44.44%	100.00%	100.00%							
12	11.80%	3.70%	2.78%	3.32%	5.15%	11.82%	25.81%	44.44%	100.00%	100.00%						
13	10.22%	2.71%	1.67%	1.90%	2.80%	5.88%	11.21%	23.53%	44.44%	100.00%	100.00%					
14	9.01%	2.03%	1.08%	1.03%	1.47%	2.57%	5.33%	10.57%	25.00%	44.44%	100.00%	100.00%				
15	7.76%	1.56%	0.71%	0.60%	0.72%	1.25%	2.19%	4.75%	10.24%	24.24%	44.44%	100.00%	100.00%			
16	7.04%	1.19%	0.49%	0.35%	0.37%	0.57%	1.00%	1.87%	4.54%	9.03%	23.53%	40.00%	100.00%	100.00%		
17	6.21%	0.95%	0.33%	0.22%	0.19%	0.27%	0.43%	0.83%	1.75%	4.12%	9.56%	21.62%	40.00%	100.00%	100.00%	
18	5.59%	0.75%	0.24%	0.13%	0.11%	0.13%	0.19%	0.35%	0.76%	1.59%	4.21%	8.44%	22.22%	40.00%	100.00%	100.00%

As Deller (2009) mentioned in his research, if an *n by n* square adjacency matrix contains no links at all, its *n* eigenvalues are all zero. If it contains a maximally connected network, one of its eigenvalues is *n*, the rest are zero. Note that the ranges of λ_{PFE}'s for the numbers of different meaningful combinations of a X-Y-X-1 networked forces are stuck in a narrow band of the full range, n, due to the relatively small differences of the links within two of 16 sub matrices. The number of discrete points within the range of eigenvalues will become insufficient for statistical analysis to explain the performance measure of a networked force.

The λ_{PFE}'s vary infinitesimally. They reflect the relationship between the probability and the combinations quite well. The λ_{PFE}'s are important measures up to around 15 Sensors and Influencers. From that point on, the numbers of unique λ_{PFE}'s over the numbers of different meaningful combinations percentage is around 1% or even less as seen from the Table 3.3, which really doesn't give anything to measure.

When the results are evaluated, it is seen that the weak BLUE configurations versus the strong RED configurations have a lower probability of a BLUE win over the equal assets of RED forces. If a Decider has only one Sensor or only one Influencer, it is very easy for that Decider to be rendered useless once its only entity is killed no matter how many other Decider the other entity has. This is the mechanism through how the agent-based modeling of the IACM works. For example, if the BLUE force with 5-1-1 Sensors vs. 1-5-1 Influencers is fighting against the RED force with 5-1-1 Sensors vs. 5-1-1, the probability of BLUE win, the actual result of the experiment, is zero.

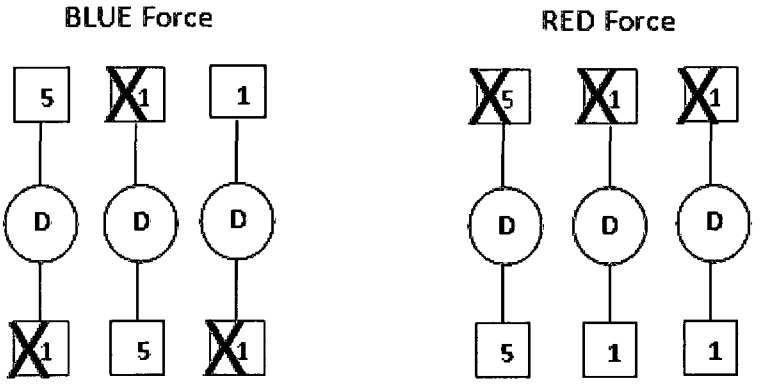


Figure 3.9. The Weakest BLUE Configuration vs the Strongest RED Configuration

In the above example, the BLUE force has the weakest configuration and the RED force has the strongest configuration. Once the only entities of each Decider are killed, the BLUE force is out of fight right away. But the RED force still has at least one Decider with enough entities linked to it that are ready for fight. There are only hierarchial links in the chain of command and no peer-to-peer links between the entities. On the other hand, the probability of a BLUE win with opposite configurations, the experiment result, is 0.967. The strength of the configuration can be defined as the number, which is greater than one, of each entity linked to each Decider (i.e. each Decider which has more than one Sensor and one Influencer is strong, the more entities linked to each Decider, the stronger the deciders and therefore the configuration will be).

Once the mechanism, that causes higher probability to win the fight, is understood, the intent is to detect how strong and determined each Decider is. In other words, give the highest weight in rank to the deciders with maximum number of Sensors and Influencers as possible and give the lowest weight in rank to the ones with one Sensor and one Influencer. That weight could be calculated by linear algebraic operations, like the max-min difference of Sensors and Influencers as ***"Disparity"***, or the summation of minimum of each Sensor-Decider pair as ***"Robustness"***. That weight could be calculated by linear matrix operations, like eigenvalues. The weight could be calculated by manipulating some functions sensitive to ones (1), like logarithmic function, or the squareroot. The logarithm and natural logarithm of one (1) is all zero. The logarithmic

functions and the squareroot are fairly sensitive to the changes in numbers the way in which to detect the strength of the connectivity in between each Deciders and its respective Sensors and Influencers.

Once the probability of BLUE win is sorted from small to large, it is easily seen that from the weakest BLUE force configuration vs. the strongest RED force configuration is at the top, and pretty much, all the way down to the opposite configuration at the bottom. No matter what metrics are used to measure the performance of the networked forces, they will vary in narrow bands (range) with increments as natural as the input of this process integer partitioning varies in narrow band.

3.4. DEVELOPING THE ANYLOGIC MODEL

The agent-based simulation environment used for this research was AnyLogic 6.4.1 University Version by Copyright (c) XJ Technologies, 1991-2009. The purpose of this section is to explain the underlying logic of key parts of the AnyLogic code used in this research; the entire code is provided in the Appendix.

The same rules as Deller (2009) used in his research were used. Sensors, Deciders, and Influencers act as agents. Targets did not serve as an agent since it acted to absorb the opposing side's losses and its representation in the X-Y-X-1 arrangement is always one as the absorbing (null) element. Target agents only serve to collect the results.

Since the Deciders are the key nodes (agents) to link multiple Sensors and Influencers, we don't want them destroyed. Deciders are immortal agents. All targets are equal importance and priority in order to generate unbiased results.

All agents placed randomly upon initiation. Once Deciders are placed, they never move. Sensors sense and detect enemy nodes within the sensing range, and pass that information to the Deciders they connected. Deciders pass the sensing information to their connected Influencers. Influencers kill the nearest assigned (directed) enemy node within the influencing range. Deciders have the

situational awareness to proact with the Sensors and Influencers to suspicious areas. All agents are assumed to perform their jobs according to the rules set forth perfectly and instantaneously. Agent-based model is built deterministically; that means whatever the agents' jobs are, their probability to be done is all 100% (Deller, 2009).

Each agent in the model is defined turtle object set of BLUE Deciders and RED Deciders with index (as being the number of Deciders). The code below is just given for BLUE Deciders to see how it works. With the same fashion, similar code is applied for RED Deciders.

```
void onChange_nBDeciders() {
        int index;
  index = 0;
  for ( Turtle object : influencersB ) {
    object.set_nFleets(nBDeciders);
    index++;
  }
  index = 0;
  for ( Turtle object : sensorsB ) {
    object.set_nFleets(nBDeciders);
    index++;
  }
}
```

Sensing range parameter is defined as sRange. Influencing range parameter is defined as iRange. Both of these parameters values are set 10 as a default value. They can be changed. For simplicity, consistent and unbiased results, they were kept as default value during the entire search space experiments. The agent-based model created in AnyLogic is so flexible that any experiment can be run by just plugging the predetermined java output X-Y-X list of configurations and changing two parameters: nBDeciders, nRDeciders. The total number of agents will be seen under the environment and each agent respectively under their names on startup in Simulation:Main. Simulation:Main just runs the experiment with the first configurations of both BLUE and RED forces with one replicate for demonstration purpose only. Once it starts, the numbers will decrease till one side's Sensors and Influencers are all killed. The numbers of Deciders stay constant; because, Deciders can't be killed as a rule described earlier.

There are functions defined to establish the hierarchial links in the chain of command in between agents (entities). These functions are "sense', "track", "shoot", "kill", "moveInfluencers", "moveSensors", and "reset".

3.4.1. SENSE FUNCTION

There are three nesting loops as shown in the code below. The first loop is DecidersB loop goes for all decidersB. The second loop is InTurtles loop. They are the attributes of the DecidersB, which are the turtles linked to DecidersB (SensorsB and InfluencersB). The third two loops are for the opposing side Targets; InfluencersR and SensorsR. If the distance from InfluencersR to inTurtles of DecidersB is less than or equal to sRange, that indexed InfluencersR is sensed. With the same fashion, if the distance from SensorsR to inTurtles of DecidersB is less than or equal to sRange, that indexed SensorsR is sensed. The same thing is also applied for the RED side in the same fashion. The sense function code is given as an example to explain how it works:

```
void
 sense( ) {

for (Turtle d: deciderB) {
        int ind = d.getIndex();
        for (Turtle s: d.inTurtles) {
                for (Turtle e: influencersR) {
                        if (s.distanceTo(e) <= sRange)
                                e.sensedBD[ind] = true;
                }
                for (Turtle e: sensorsR) {
                        if (s.distanceTo(e) <= sRange)
                                e.sensedBD[ind] = true;
                }
        }
}

for (Turtle d: deciderR) {
        int ind = d.getIndex();
        for (Turtle s: d.inTurtles) {
                for (Turtle e: influencersB) {
                        if (s.distanceTo(e) <= sRange)
                                e.sensedRD[ind] = true;
                }
                for (Turtle e: sensorsB) {
                        if (s.distanceTo(e) <= sRange)
                                e.sensedRD[ind] = true;
                }
```

```
        }
    }
}
```

3.4.2. TRACK FUNCTION

This function just shows the tracking links. There are also three nesting loops as shown in the code below. The first loop is DecidersB loop goes for all DecidersB. The second loop is OutTurtles loop. They are the attributes of the DecidersB, which are the turtles linked to DecidersB (SensorsB and InfluencersB). The third two loops are for the opposing side Targets; InfluencersR and SensorsR. If the distance from InfluencersR to OutTurtles of DecidersB is less than or equal to iRange, that InfluencersR is added to outTurtles list and tracked. In the same fashion, if the distance from SensorsR to outTurtles of DecidersB is less than or equal to iRange, that SensorsR is added to outTurtle list and tracked. The same thing is also applied for the RED side with the same fashion. The track function code is given as an example to explain how it works:

```
void
 track( ) {

for (Turtle d: deciderB) {
        for (Turtle s: d.outTurtles) {
                for (Turtle e: influencersR) {
                        if (s.distanceTo(e) <= iRange)
                                s.outTurtles.add(e);
                }
                for (Turtle e: sensorsR) {
                        if (s.distanceTo(e) <= iRange)
                                s.outTurtles.add(e);
                }
        }
}

for (Turtle d: deciderR) {
        for (Turtle s: d.outTurtles) {
                for (Turtle e: influencersB) {
                        if (s.distanceTo(e) <= iRange)
                                s.outTurtles.add(e);
                }
                for (Turtle e: sensorsB) {
                        if (s.distanceTo(e) <= iRange)
                                s.outTurtles.add(e);
                }
         }
```

```
            }
        }
```

3.4.3. SHOOT FUNCTION

There are also three nesting loops and three new variables defined here. These variables are closestTarget, closestDistance and dist. The closestTarget is defined as turtle and initiated as null. The closestdDistance defined as double variable and initiated as positive infinity. The dist is defined as distance from possible targets to outTurtles of DecidersB. Turtle e, defined as outTurtles attribute of turtle s of outTurtles of DecidersB, if not sensed, if dist is less than positive infinity (dist is definitely less), then that turtle e is closest target and the dist is the closestDistance. If closestTarget is not null, then closestTarget is dead. Likewise, the RED shooting function is explained in the code below:

```
void
 shoot( ) {

for (Turtle d: deciderB) {
        int ind = d.getIndex();
        for (Turtle s: d.outTurtles) {
                Turtle closestTarget = null;
                double closestDistance = Double.POSITIVE_INFINITY;
                for (Turtle e: s.outTurtles) {
                        if (!e.sensedBD[ind]) {
                                continue;
                        }
                        double dist = s.distanceTo(e);
                        if (dist < closestDistance) {
                                closestTarget = e;
                                closestDistance = dist;
                        }
                }
                if (closestTarget != null) {
                        closestTarget.dead = 1;

                }
        }
}
for (Turtle d: deciderR) {
        int ind = d.getIndex();
        for (Turtle s: d.outTurtles) {
                Turtle closestTarget = null;
                double closestDistance = Double.POSITIVE_INFINITY;
                for (Turtle e: s.outTurtles) {
                        if (!e.sensedRD[ind])
                                continue;
                        double dist = s.distanceTo(e);
```

```
                        if (dist < closestDistance) {
                                closestTarget = e;
                                closestDistance = dist;
                        }

                }
                if (closestTarget != null) {
                        closestTarget.dead = 1;
;
                }
        }
    }
  }
```

3.4.4. KILL FUNCTION

There are four loops for every possible target from each side. The possible targets are InfluencersR, InfluencersB, SensorsR, and SensorsB. The loops go for their sizes and check if their attribute "dead' equals to 1. If they are dead, they are removed the each agent list as seen in the code below:

```
void
 kill( ) {

for (int i = influencersR.size()-1; i>=0; i--) {
        Turtle t = influencersR.get(i);
        if (t.dead == 1) {
                remove_influencersR(t);
        }
}
for (int i = influencersB.size()-1; i>=0; i--) {
        Turtle t = influencersB.get(i);
        if (t.dead == 1) {
                remove_influencersB(t);
        }
}
for (int i = sensorsR.size()-1; i>=0; i--) {
        Turtle t = sensorsR.get(i);
        if (t.dead == 1) {
                remove_sensorsR(t);
        }
}

for (int i = sensorsB.size()-1; i>=0; i--) {
        Turtle t = sensorsB.get(i);
        if (t.dead == 1) {
                remove_sensorsB(t);
        }
}

}
```

3.4.5. MOVEINFLUENCERS FUNCTION

There are three nesting loops. The outer loop goes for all Decidersb. The second loop goes for all indexed outTurtles of DecidersB. There are two variables defined and initiated; closestTarget as turtle and it is null, closestDistance as double variable and it is positive infinity. In the inner loop, for every InfluencersR, if they are not sensed or dead, continue, if the distance from each InfluencersR to outTurtles of DecidersB is less than positive infinity (it is obviously less than infinity) and the same thing applied for the SensorsR. Then if closestTarget is not null, move the InfluencersB to a calculated i.set XY coordinates as in the code below. The same thing is applied for the RED side in the same fashion.

```
void
 moveInfluencers( ) {

for (Turtle d: deciderB) {
        int ind = d.getIndex();
        for (Turtle i: d.outTurtles) {
                Turtle closestTarget = null;
                double closestDistance = Double.POSITIVE_INFINITY;
                for (Turtle e: influencersR) {
                        if (!e.sensedBD[ind] || e.dead == 1)
                                continue;
                        double dist = i.distanceTo(e);
                        if (dist < closestDistance) {
                                closestTarget = e;
                                closestDistance = dist;
                        }
                }
                for (Turtle e: sensorsR) {
                        if (!e.sensedBD[ind] || e.dead == 1)
                                continue;
                        double dist = i.distanceTo(e);
                        if (dist < closestDistance) {
                                closestTarget = e;
                                closestDistance = dist;
                        }
                }
                // move
                if (closestTarget != null) {
                        i.setXY(i.getX() + (closestTarget.getX() -
i.getX())/closestDistance , i.getY() + (closestTarget.getY() - i.getY())/closestDistance);
                }
        }
}
```

moveSensors Function: There are two separate two nesting loops. The first loop goes for all DecidersB. Within the first loop, for every InfluencersR and SensorsR, if they are not sensed and are not dead, sense them. If they are sensed, continue.

The second loop goes for all inTurtles of DecidersB.

4. MODELING RESULTS

The search space was limited to a reasonable numbers due to the enormous computational requirements as the number of different meaningful combinations grew exponentially. The experiments started from three Deciders and Sensors-Influencers to 12 Deciders and Sensors-Influencers as shown in Table 4.1. A total of 55 experiments were conducted in this research. Each experiment consisted of all possible force-on-force engagements of the number of different meaningful combinations of two networked forces (BLUE and RED) containing X Sensors, Y Deciders, X Influencers, and one (1) Target. The sole Target node represents all the possible enemy nodes vulnerable to being targeted and it clusters the hit enemy nodes.

Table 4.1. The Numbers of Different Meaningful Combinations of all X-Y-X-1 Networked Forces where X<13 and Y<13

		Number of Deciders (Y)									
Numbers of Sensors (X) and Influencers (X)		**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	3	1									
	4	2	1								
	5	8	2	1							
	6	19	9	2	1						
	7	42	27	9	2	1					
	8	78	74	30	9	2	1				
	9	139	168	95	31	9	2	1			
	10	224	363	248	105	31	9	2	1		
	11	350	703	614	301	108	32	9	2	1	
	12	517	1297	1367	814	325	110	31	9	2	1

Each side has equal assets of force with identical capabilities for similar nodes. Since each side has exactly the same number of nodes, then, the outcome of the experiments most likely reflects the result of how variously they are connected to the IACM structure. A comprehensive test of each combination against each other requires so many engagements as the square of the number of different meaningful combinations. For normally and random distribution of both sides nodes across the battlespace, each engagement replicates 30 times.

The number of different meaningful combinations versus the same number times 30 replications of iterations are run for each case. Every iteration might result in one of the following; a BLUE win, a RED Win, or an undecided result (no winner).

The probability of each BLUE combination win against for all RED combinations was calculated as the percentage of that particular BLUE combination Wins within the number of all different meaningful RED combinations of 30 replicates.

$$P(BLUEWin)_i = \frac{\sum_{i=1}^{m} BWin_{ijk} = 1}{n * p}$$

The probability of each BLUE combination win against each RED combination was calculated as the percentage of that particular BLUE combination Wins versus the same RED combination of 30 replicates.

$$P(BLUEWin)_{ij} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (BWin_{ijk} = 1)}{p}$$

Where, i is the number of different meaningful BLUE combinations, $1 \leq i \leq m$

j is the number of different meaningful RED combinations, $1 \leq j \leq n$

k is the number of replicates, $1 \leq k \leq p=30$

This chapter was split into three sections. The first section gives the definition of each metric that will be used to measure the performance of a networked force.

The second section investigates each BLUE combination versus all RED combinations performance of all 55 experiments aggregated data and each individual experiment data with respect to metrics used before and metrics proposed in this research.

The third section investigates each BLUE combination versus each RED combination performance of all 55 experiments aggregated data and each individual experiment data in the same fashion.

4.1. DEFINITION OF EACH METRICS.

4.1.1. EIGENVALUES

They are a special set of scalars associated with a linear system of equations that are also known as characteristic roots, characteristic values, proper values, or latent roots. The determination of the eigenvalues and eigenvectors of a system is extremely important in physics and engineering to explain the characteristic behavior of a system. Therefore, eigenvalues are used in this research to explain the performance of a networked force.

The greater the eigenvalue of a combination, the grater the likelihood of a high value for probability to win.

4.1.2. DISPARITY

It is the sum of the max-min difference of Sensors and Influencers across the Deciders. This can be formulated as (Deller 2009):

$$Disparity = [\max(S_n) - \min(S_n)] + [\max(I_n) - \min(I_n)]$$

Where, S_n: the number of Sensors assigned to each of n Deciders

I_n : the number of Influencers assigned to each of n Deciders

The greater disparity most likely creates either an extremely high or low value for probability to win.

4.1.3. ROBUSTNESS

It is the minimum number of either Sensors or Influencers lost that would render all the Deciders and the rest of the nodes nonfunctional. This can be formulated as :

$$Robustness = \sum_{i=1}^{n} \min(S_i, I_i)$$

Where, S_i : the number of Sensors assigned to Decider *i*

I_i : the number of Influencers assigned to Deciders *i*

The greater the robustness value, most likely the larger the probability to win the fight. The higher robustness value reflects how Sensor-Decider-Influencer fighting triad strongly connected to one another in the IACM structure to maintain the combat effectiveness.

4.1.4. STRENGTH

One of the proposed metrics in this research is "***the strength of connectivity***". For simplicity, it is called as "Strength".

It is the sum of weighted average according to the logarithmic function of each Decider and so the combination that reflects how many nodes of Sensors and Influencers linked to each Decider so that the entire combination maintains the combat effectiveness. This can be formulated as :

$$\text{Strength} = \sum_{i=1}^{n} \{\log_{10}(\#\text{ of Sensor}_i + 1) * \log_{10}(\#\text{ of Influencer}_i + 1)\}$$

To clarify the rationale, the logarithmic values of some numbers and the strength of a configuration are given below in Table 4.2;

Table 4.2. A Sample Strength Calculation

Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Log_{10}	0	0.301	0.477	0.602	0.699	0.778	0.845	0.903	0.954	1.000	1.041	1.079	1.114

# of BSen1s	# of BSen2s	# of BSen3s	
5	1	1	Strength
1	5	1	0.210411
# of BInf1s	# of BInf2s	# of BInf3s	

versus

# of RSen1s	# of RSen2s	# of RSen3s	
5	1	1	Strength
5	1	1	0.4885591
# of RInf1s	# of RInf2s	# of RInf3s	

As seen in the above configuration, the BLUE force has the weakest configuration and the RED force has the strongest configuration and the strength

varies in between zero and 0.4885591. The strength values are confined to a narrow range ([0.210411 - 0.4885591]) as in the eigenvalues.

The greater the strength value, most likely the larger the probability to win the fight just like the other metrics except disparity.

4.1.5. POWER

Another proposed metric in this research is "*the power of the Deciders*". It is also called as "Power".

It is also another sum of weighted average according to the squareroot function of each decider and so the combination that reflects how many nodes of Sensors and Influencers linked to each Decider so that the entire combination maintains the combat effectiveness. This can be formulated as :

$$\text{Power} = \sum_{i=1}^{n} \{\text{Sqrt}(\#\ \text{of Sensor}_i) * \text{Sqrt}(\#\ \text{of Influencer}_i)\}$$

To clarify the rationale, the squareroot values of some numbers and the power of a configuration are given below in Table 4.3;

Table 4.3. A Sample Power Calculation

Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Sqrt	1	1.414	1.732	2.000	2.236	2.449	2.646	2.828	3.000	3.162	3.317	3.464	3.606

# of BSen1s	# of BSen2s	# of BSen3s	
5	1	1	Power
1	5	1	5.472136
# of BInf1s	# of BInf2s	# of BInf3s	

versus

# of BSen1s	# of BSen2s	# of BSen3s	
5	1	1	Power
5	1	1	7
# of BInf1s	# of BInf2s	# of BInf3s	

This time, the power range varies in between 5.4721 to 7.

The larger the power value, the more reliable and readily available fighting units maintains the combat effectiveness.

4.1.6. STABILITY

Another proposed metric in this research is "Stability of Deciders", referred to as Stability.

It is the sum of quotient of Sensors and Influencers connected to each Decider and it can be describes as:

$$Stability = \sum_{i=1}^{n} \{\text{Quotient}(\#\ \text{of Sensor}_i, \#\ \text{of Influencer}_i)\}$$

There is a negative correlation in between the combat performance and the stability value. It shows the number of ineffectively used Decider nodes.

4.1.7. CONNECTIVITY

The last metric prosed in this research is "Connectivity of Sensors/Influencers", referred to as Connectivity.

It is the sum of unbalanced absolute number of Sensors and Influencers of the Deciders.

$$Connectivity = \sum_{i=1}^{n} \{\text{ABS}(\#\ \text{of Sensor}_i) - (\#\ \text{of Influencer}_i)\}$$

There is a fair degree of negative correlation between the combat performance and the connectivity value. It represents the number of unproductive Sensors/Influencers.

4.2. PERFORMANCE OF EACH BLUE COMBINATION VS ALL RED COMBINATIONS

Each BLUE combination vs. all RED combinations respective of all 55 experiments have a total number of 8,340 datasets. These datasets contain the probability of each BLUE combination win (dependent variable) versus all RED combinations, and metrics such as eigenvalue, disparity, robustness, power of Deciders, strength of connectivity, and stability of Deciders. In this section, the

probability of BLUE win for its each combination is studied for all combinations of the RED side. They are run in the SPSS Statistics 17.0 software package.

4.2.1. THE ANALYSIS OF EXPERIMENTS WITH RESPECT TO EIGENVALUE

4.2.1.1. The Analysis of All Experiments With Respect To Eigenvalue

When Table 4.4 is examined, the eigenvalues are not a good predictor or performance metric by itself alone for a networked force. It must be enhanced by some other metrics to measure the performance or predict the probability of win of a networked force.

Table 4.4. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue

Descriptive Statistics

	Mean	Std. Deviation	N
P(BWin)	.48	.119	8340
BLUE_Eigenvalues	2.29	.207	8340

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.580[a]	.336	.336	.097	.336	4224.768	1	8338	.000

a. Predictors: (Constant), BLUE_Eigenvalues

b. Dependent Variable: P(BWin)

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B	
		B	Std. Error	Beta			Lower Bound	Upper Bound
1	(Constant)	-.289	.012		-24.412	.000	-.313	-.266
	BLUE_Eigenvalues	.335	.005	.580	64.998	.000	.325	.345

a. Dependent Variable: P(BWin)

The multiple correlation coefficient, R, is the linear correlation between the observed and model predicted values of the probability of a BLUE win. Its value is 58% which indicates a moderate relationship.

The coefficient of determination, R Square (R^2), is the squared value of the multiple correlation coefficient. It shows that about 33.6% of the variation in probability of a BLUE win is explained by the model, which is very low.

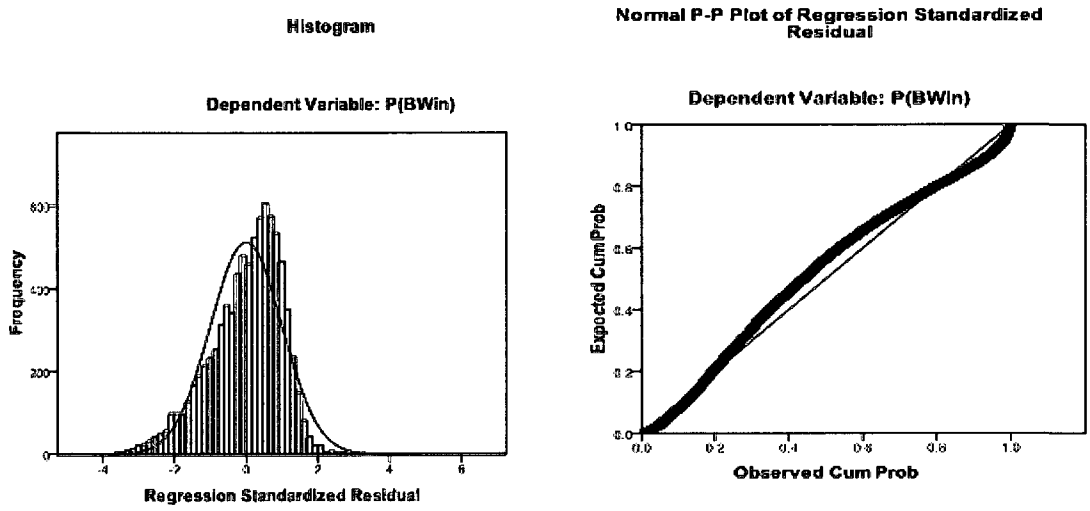


Figure 4.1. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue

The results of the linear regression yield the following equation:

$$y = 0.335x - 0.289$$

Where, y: the average probability of a BLUE win for that configuration

x: the λ_{PFE} value of a configuration

4.2.1.2. The Analysis of Each Experiment With Respect To Eigenvalue

Table 4.5. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	1	1			
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.869	.755	.714	.040	.005
Model5.4.5	1	1			
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.912	.832	.822	.043	.000
Model6.4.6	.933	.871	.852	.032	.000
Model6.5.6	1	1			
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.932	.868	.865	.042	.000
Model7.4.7	.935	.874	.868	.027	.000
Model7.5.7	.901	.812	.785	.030	.001
Model7.6.7	1	1			
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.936	.877	.875	.043	.000
Model8.4.8	.913	.833	.831	.040	.000
Model8.5.8	.862	.743	.733	.037	.000
Model8.6.8	.884	.782	.751	.047	.002
Model8.7.8	1	1			
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.931	.866	.865	.049	.000
Model9.4.9	.925	.855	.854	.040	.000
Model9.5.9	.903	.815	.813	.038	.000
Model9.6.9	.871	.759	.751	.038	.000
Model9.7.9	.830	.688	.644	.039	.006
Model9.8.9	1	1			
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				
Model10.3.10	.933	.870	.870	.052	.000
Model10.4.10	.924	.854	.853	.044	.000
Model10.5.10	.903	.815	.815	.041	.000
Model10.6.10	.892	.797	.795	.037	.000
Model10.7.10	.840	.706	.696	.031	.000
Model10.8.10	.563	.317	.220	.034	.114
Model10.9.10	1	1			
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				

Table 4.5. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model11.3.11	.932	.868	.868	.056	.000
Model11.4.11	.918	.843	.843	.051	.000
Model11.5.11	.905	.819	.819	.046	.000
Model11.6.11	.874	.764	.763	.043	.000
Model11.7.11	.866	.750	.747	.038	.000
Model11.8.11	.739	.546	.531	.036	.000
Model11.9.11	.425	.181	.064	.034	.254
Model11.10.11	1	1			
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.912	.833	.832	.067	.000
Model12.4.12	.922	.851	.851	.053	.000
Model12.5.12	.892	.795	.795	.053	.000
Model12.6.12	.893	.798	.798	.043	.000
Model12.7.12	.869	.756	.755	.041	.000
Model12.8.12	.887	.787	.785	.036	.000
Model12.9.12	.752	.565	.550	.034	.000
Model12.10.12	.755	.570	.509	.036	.019
Model12.11.12	1	1			
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

When the individual experiment results calculated by just the eigenvalues are examined in Table 4.5, above, the models with large number of Sensors and Influencers with respect to low number of Deciders have higher R and R square values. The experiments with two iterations have R and R square value of one; a perfect regression line needs only two points. When the difference between the number of Sensors/Influencers and Deciders get closer to each other, the R and R square values drop dramatically, then the experiments become insignificant.

4.2.1.3. The Analysis of Decider Basis Experiments With Respect To Eigenvalue

Table 4.6. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.628	.394	.394	.117	.000
ModelX4X	.672	.452	.452	.095	.000
ModelX5X	.659	.434	.434	.083	.000
ModelX6X	.715	.511	.511	.064	.000
ModelX7X	.676	.457	.456	.059	.000
ModelX8X	.716	.512	.509	.051	.000
ModelX9X	.531	.282	.265	.044	.000
ModelX10X	.162	.026	-.071	.082	.614
ModelX11X	.752	.565	.130	.078	.459
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by just the eigenvalues along with the Deciders (column-wise) are examined in Table 4.6, above, the R and R square values are quiet low. Moreover, the experiments get insignificant as the number of Deciders increases and the number of total iterations decreases. The experiments with 10 Deciders and further are insignificant.

4.2.1.4. The Analysis of Sensor/Influencer Basis Experiments With Respect To Eigenvalue

Table 4.7. Collective Regression Results for the Sensor/Influencer Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	.865	.748	.496	.095	.335
Model5Y5	.547	.299	.221	.061	.082
Model6Y6	.480	.230	.204	.083	.006
Model7Y7	.564	.318	.309	.079	.000
Model8Y8	.613	.375	.372	.081	.000
Model9Y9	.646	.418	.416	.084	.000
Model10Y10	.648	.420	.419	.086	.000
Model11Y11	.645	.416	.416	.092	.000
Model12Y12	.650	.422	.422	.093	.000

When the experiments results are calculated by applying just the eigenvalues along with the Sensors/Influencers (row-wise) (examined in Table

4.7 above), the R and R square values are also quiet low but they are stable. The value of Rs stays in the mid-60's percentage-wise, and the value of R squares stays low-in the 40's percentage-wise. The experiments start insignificant initially due to low number of total iterations, as the number of Sensors/Influencers increases so does and the sum of total iterations, they become significant after 5 Senosrs/Influencers.

4.2.2. THE ANALYSIS OF EXPERIMENT WITH RESPECT TO EIGENVALUE, DISPARITY, AND ROBUSTNESS

4.2.2.1. The Analysis of All Experiments With Respect To Eigenvalue, Disparity, and Robustness

Table 4.8. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Robustness

Descriptive Statistics

	Mean	Std. Deviation	N
PBWin	.48	.119	8340
BLUE_Eigenvalues	2.29	.207	8340
BLUE_TotalDisparity	7.22	2.555	8340
BLUE_Robustness	7.71	1.737	8340

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.891[a]	.794	.794	.054	.794	10585.395	3	8336	.000

a. Predictors: (Constant), BLUE_Robustness, BLUE_TotalDisparity, BLUE_Eigenvalues

b. Dependent Variable: PBWin

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B	
		B	Std. Error	Beta			Lower Bound	Upper Bound
1	(Constant)	-.354	.008		-48.294	.000	-.378	-.349
	BLUE_Eigenvalues	.320	.005	.554	60.980	.000	.310	.330
	BLUE_TotalDisparity	-.018	.000	-.393	-48.033	.000	-.019	-.018
	BLUE_Robustness	.031	.001	.455	55.135	.000	.030	.032

a. Dependent Variable: PBWin

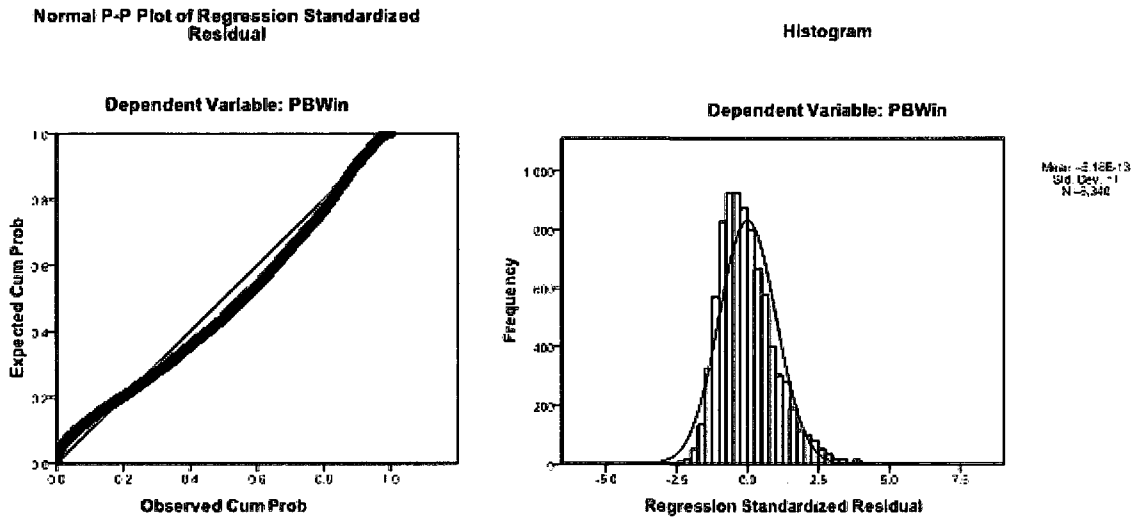


Figure 4.2. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Robustness

When the experiment results of collected 8,340 datasets were calculated by applying eigenvalues, disparities, and robustnesses and are examined in the Table 4.8 above. A regression analysis of the λ_{PFE}, the disparity, and the robustness values yields a tremendous increase in the coefficient of determination, R square (R^2) from a value of 0.336 to 0.794 and provides the following equation:

$$y = 0.320x_1 - 0.018x_2 + 0.031x_3 - 0.364$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of a configuration

x_2: the disparity value of a configuration

x_3: the robustness value of a configuration

4.2.2.2. The Analysis of Each Experiment With Respect To Eigenvalue, Disparity, and Robustness

Table 4.9. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	1	1			
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.911	.829	.701	.040	.051
Model5.4.5	1	1			
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.965	.931	.918	.029	.000*
Model6.4.6	.961	.924	.879	.029	.003*
Model6.5.6	1	1			
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.987	.974	.972	.019	.000
Model7.4.7	.969	.939	.931	.020	.000*
Model7.5.7	.929	.863	.780	.031	.014*
Model7.6.7	1	1			
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.987	.975	.974	.019	.000
Model8.4.8	.976	.953	.951	.021	.000
Model8.5.8	.956	.914	.904	.022	.000**
Model8.6.8	.948	.899	.838	.038	.006*
Model8.7.8	1	1			
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.991	.982	.981	.018	.000
Model9.4.9	.988	.976	.976	.017	.000
Model9.5.9	.979	.959	.958	.018	.000
Model9.6.9	.974	.950	.944	.018	.000**
Model9.7.9	.982	.964	.943	.016	.000**
Model9.8.9	1	1			
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				
Model10.3.10	.993	.987	.986	.017	.000
Model10.4.10	.989	.978	.978	.017	.000
Model10.5.10	.982	.964	.964	.018	.000
Model10.6.10	.972	.944	.942	.020	.000
Model10.7.10	.953	.909	.899	.018	.000
Model10.8.10	.707	.500	.200	.035	.288
Model10.9.10	1	1			
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				

Table 4.9. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model11.3.11	.992	.985	.984	.019	.000
Model11.4.11	.991	.982	.982	.018	.000
Model11.5.11	.988	.975	.975	.017	.000
Model11.6.11	.985	.971	.971	.015	.000
Model11.7.11	.966	.933	.931	.020	.000
Model11.8.11	.892	.796	.773	.025	.000**
Model11.9.11	.694	.482	.171	.032	.311
Model11.10.11	1	1			
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.992	.983	.983	.021	.000
Model12.4.12	.992	.984	.984	.018	.000
Model12.5.12	.989	.977	.977	.018	.000
Model12.6.12	.985	.971	.971	.016	.000
Model12.7.12	.982	.965	.965	.016	.000
Model12.8.12	.971	.943	.941	.019	.000
Model12.9.12	.866	.750	.722	.026	.000*
Model12.10.12	.962	.925	.881	.018	.003*
Model12.11.12	1	1			
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

When the individual experiment results calculated by applying eigenvalues, disparities, and robustnesses are examined in Table 4.9 above, the models with large number of Sensors and Influencers with respect to low number of Deciders have higher R and R square values. When the difference between the number of Sensors/Influencers and Deciders gets closer to each other, the number of different meaningful combinaions and thereby the number of iterations drops. So R and R square values drop dramatically as a consequence of this; then the experiments become insignificant. The models that have less than or equal to 30 number of iterations (i.e., the sample size, the number of different meaningful combinations) are insignificant. The models with one or two asterisks in the significant column are insignificant models due to individual insignificance in its independent variables even if they look significant as a whole model.

4.2.2.3. The Analysis of Decider Basis Experiments With Respect To Eigenvalue, Total Disparity, and Robustness

Table 4.10. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.907	.823	.823	.063	.000
ModelX4X	.905	.818	.818	.055	.000
ModelX5X	.903	.815	.814	.048	.000
ModelX6X	.923	.853	.852	.035	.000
ModelX7X	.912	.832	.831	.033	.000
ModelX8X	.908	.825	.821	.031	.000
ModelX9X	.763	.582	.550	.034	.000*
ModelX10X	.610	.372	.137	.073	.268
ModelX11X	1	1			
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by applying eigenvalues, disparities, and robustnesses along with the deciders (column-wise) are examined in Table 4.10, above, the R and R square values are at least 50% higher than the results calculated by just applying eigenvalues. The experiments with 9 Deciders and further are insignificant. The models with 9 Deciders looks significant as a whole model. But indeed, it is an insignificant model from an individual independent variables perpective.

4.2.2.4. The Analysis of Sensor/Influencer Basis Experiments With Respect To Eigenvalue, Total Disparity, and Robustness

Table 4.11. Collective Regression Results for the Sensor/Influencer Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	1	1			
Model5Y5	.894	.798	.712	.037	.008*
Model6Y6	.937	.878	.864	.034	.000*
Model7Y7	.953	.908	.904	.029	.000
Model8Y8	.950	.903	.902	.032	.000
Model9Y9	.956	.914	.913	.032	.000
Model10Y10	.959	.920	.920	.032	.000
Model11Y11	.964	.930	.930	.032	.000
Model12Y12	.972	.945	.945	.029	.000

When the experiments results calculated by applying eigenvalues, disparities and robustnesses along with the sensors/influencers (row-wise) are examined in Table 4.11, above, the R and R square values are so much better than the results calculated by just applying eigenvalues. The value of Rs increases up to the mid 90's percent, and the value of R squares increases up to the 90's percent. The experiments start initially insignificant due to small number of total iterations, as the number of Sensors/Influencers increases so does and the sum of total iterations, they become significant after 6 Senosrs/Influencers.

4.2.3. THE ANALYSIS OF ALL EXPERIMENTS WITH RESPECT TO EIGENVALUE, DISPARITY, ROBUSTNESS, POWER, AND CONNECTIVITY

Table 4.12. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, Robustness, Power, and Connectivity

Descriptive Statistics

	Mean	Std. Deviation	N
PBWin	.48	.119	8340
BLUE_Eigenvalues	2.29	.207	8340
BLUE_TotalDisparity	7.22	2.555	8340
BLUE_Robustness	7.71	1.737	8340
BLUE_Power	10.27	1.231	8340
BLUE_Connectivity	6.93	3.322	8340

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.983[a]	.966	.966	.022	.966	46789.119	5	8334	.000

a. Predictors: (Constant), BLUE_Connectivity, BLUE_Eigenvalues, BLUE_Power, BLUE_TotalDisparity, BLUE_Robustness

b. Dependent Variable: PBWin

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B	
		B	Std. Error	Beta			Lower Bound	Upper Bound
1	(Constant)	.161	.004		39.258	.000	.153	.169
	BLUE_Eigenvalues	.166	.003	.288	63.561	.000	.161	.172
	BLUE_TotalDisparity	.003	.000	.071	15.977	.000	.003	.004
	BLUE_Robustness	-.118	.001	-1.721	-84.137	.000	-.121	-.116
	BLUE_Power	.124	.001	1.283	87.617	.000	.122	.127
	BLUE_Connectivity	-.066	.000	-1.823	-132.021	.000	-.066	-.065

a. Dependent Variable: PBWin

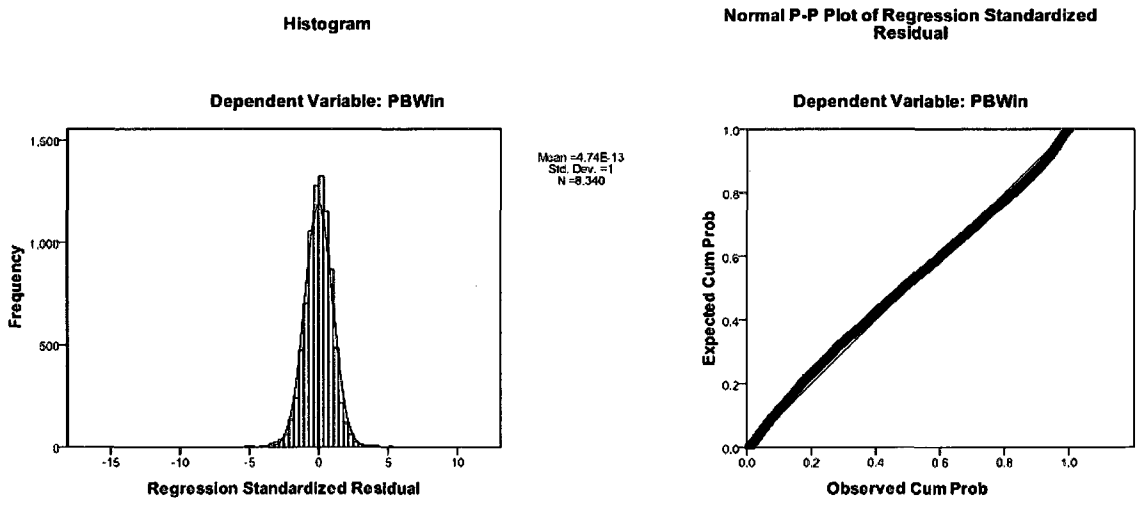


Figure 4.3. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, Robustness, Power, and Connectivity

When the experiment result of the collected 8,340 datasets calculated by applying eigenvalue, disparity, robustness, power, and connectivity are examined in the Table 4.12 above, a regression analysis of the λ_{PFE}, the disparity, the robustness, the power, and the connectivity values yields a significant increase in the coefficient of determination, R square (R^2) from a value of 0.794 to 0.966 and provides the following equation:

$$y = 0.161 + 0.166x_1 + 0.003x_2 - 0.118x_3 + 0.124x_4 - 0.066x_5$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of a configuration

x_2: the disparity value of a configuration

x_3: the robustness value of a configuration

x_4: the power value of a configuration

x_5: the connectivity value of a configuration

Since the overall R^2 value is high, and the corresponding P value is zero, the model fits the data well. The independent variables used in the regression analysis have a significant impact on the model.

Hovewer, substraction of strength and stability values from the regression analysis gives exactly the same result even though they have lower P values and small coefficients. Then it can easily be said that these two independent variables are redundant. The strength and the power values are highly correlated (0.954); they both convey essentially the same information. The stability value is moderately corelated with the connectivity value (0.495) and the eigenvalue (0.558): both the connectivity and the eigenvalue convey fairly the same information as the stability does. Each independent variable is derived from the structure of different meaningful combinations by applying various operations as described earlier.

4.2.3.1. Multicolinearity

There is a perfect linear relationship among the independent variables since R and R^2 values are very high. When there is a perfect linear relationship among the independent variables, the estimates for the model can be computed in several ways.

When a regression analysis is applied to each experiment by using the eigenvalue, the disparity, the robustness, the power, the strength, the connectivity and the stability, there seems to be a good linear relationship among the independent variables since R and R^2 are still high and the overall P is very low. Even though the overall P value is very low, all of the individual P values are high. This means that the model fits the data well, even though none of the independent variables have a statistically significant impact on predicting the probability of a BLUE win. This relation is called multicolinearity or ill conditioning (Alin, 2010). Colinearity refers to the linear relationship among two variables while multicolinearity does more variables, which also means lack of orthogonality among them.

The goal of this research is to understand how the various metrics (independent variables) impact the performance of a networked force. For that reason, multicolinearity is a big problem to solve. One problem is that the

individual P values can be misleading (a P value can be high, even though the variable is important). The second problem is that the confidence intervals on the regression coefficients will be very wide. This will cause another problem: excluding an independent variable (or adding a new one) can change the coefficients dramatically – may even change their signs.

4.2.3.2. What Can Be Done About Multicolinearity

The best solution is to find a way to understand what causes the multicolinearity and remove it. Multicolinearity occurs when two or more variables are related. They measure essentially the same thing. If one of the variables does not seem logically essential to the model, removing it may reduce or eliminate multicolinearity. The impact of multicolinearity can also be reduced by increasing the sample size. That way confidence intervals get narrower, despite multicolinearity, with more data.

The regression analysis of the model 8.5.8 is given as an example to explain the multicolinearity in three steps.

Table 4.13. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT all Metrics (Multicolinearity Analysis-Step 1)

Descriptive Statistics

	Mean	Std. Deviation	N
PBWin	.48	.071	30
BLUE_Eigenvalues	1.91	.075	30
BLUESensors_Disparity	1.93	.740	30
BLUEInfluencers_Disparity	1.93	.740	30
BLUE_TotalDisparity	3.87	1.074	30
BLUE_Robustness	6.17	.950	30
BLUE_Power	7.60	.277	30
BLUE_Strength	.79	.047	30
BLUE_Connectivity	3.67	1.900	30
BLUE_Stability	5.27	.868	30

Model Summary[b]

					Change Statistics				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	.958[a]	.918	.892	.023	.918	35.317	7	22	.000

a. Predictors: (Constant), BLUE_Stability, BLUESensors_Disparity, BLUE_Eigenvalues, BLUEInfluencers_Disparity, BLUE_Connectivity, BLUE_Strength, BLUE_Power

b. Dependent Variable: PBWin

Coefficients[a]

		Unstandardized Coefficients		Standardized Coefficients			95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
Model		B	Std. Error	Beta	t	Sig.	Lower Bound	Upper Bound	Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	-2.026	2.442		-.830	.416	-7.091	3.039					
	BLUE_Eigenvalues	.497	.935	.524	.532	.600	-1.442	2.435	.862	.113	.032	.004	281.655
	BLUESensors_Disparity	-.023	.037	-.236	-.605	.551	-.100	.055	-.111	-.128	-.037	.024	40.942
	BLUEInfluencers_Disparity	-.003	.034	-.030	-.086	.933	-.072	.067	-.112	-.018	-.005	.030	32.904
	BLUE_Power	.480	.937	1.876	.513	.613	-1.462	2.423	.938	.109	.031	.000	3602.538
	BLUE_Strength	-2.447	4.400	-1.611	-.556	.584	-11.573	6.678	.928	-.118	-.034	.000	2259.475
	BLUE_Connectivity	.000	.016	-.005	-.011	.991	-.034	.033	-.861	-.002	.000	.020	50.483
	BLUE_Stability	-.020	.028	-.243	-.713	.483	-.078	.038	-.693	-.150	-.043	.032	31.203

a. Dependent Variable: PBWin

When the Table 4.13 above is examined carefully, the model has a perfect linear relationship among the independent variables since R and R^2 values are very high and the overall P is very low; but all of the individual P values are high.

There are two values displayed in the the colinearity statistics column for each variable as a check for multicolinearity: tolerance and variance inflation factor "VIF". The tolerance is an indication of the percent of variance in the independent variable that cannot be accounted for by the other variables; hence very small values indicate that a variable is redundant, and values that are less than 0.10 may merit further investigation. The VIF is inversely proportional to the tolerance and as a rule of thumb, a variable whose VIF values is greater than 10 may merit further investigation.

All variables have less than 0.10 value in tolerance. The numbers in the tolerance column indicate that only 0.4, 2.4, 3, 0, 0, 2, and 3.2% of the variance in respective independent variables are not predictable given the other variables in the model. All of these variables measure probability of BLUE win and the very low "tolerance" values indicate that these variables contain redundant information. Multicolinearity arises because too many variables have been put in that measure the same thing, probability of BLUE win.

When the BLUE_Power and BLUE_Strength with zero in tolerance value are omited from the regression model, the new VIF values in the analysis in Table 4.14 below appears much better, but it still needs some work.

Table 4.14. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT all Metrics (Multicolinearity Analysis-Step 2)

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.958[a]	.917	.900	.022	.917	53.109	5	24	.000

a. Predictors: (Constant), BLUE_Stability, BLUESensors_Disparity, BLUE_Eigenvalues, BLUEInfluencers_Disparity, BLUE_Connectivity

b. Dependent Variable: PBWin

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Lower Bound	Upper Bound	Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	-1.053	.579		-1.817	.082	-2.248	.143					
	BLUE_Eigenvalues	.887	.275	.936	3.226	.004	.320	1.455	.862	.550	.190	.041	24.366
	BLUESensors_Disparity	-.032	.007	-.330	-4.454	.000	-.046	-.017	-.111	-.673	-.262	.628	1.592
	BLUEInfluencers_Disparity	-.017	.018	-.176	-.931	.361	-.054	.021	-.112	-.187	-.055	.097	10.355
	BLUE_Connectivity	.003	.007	.083	.425	.675	-.012	.018	-.861	.086	.025	.090	11.146
	BLUE_Stability	-.015	.016	-.182	-.954	.350	-.047	.017	-.693	-.191	-.056	.095	10.520

a. Dependent Variable: PBWin

In Table 4.14 above, there are four out of five variables that have less than 0.10 value in tolerance. The BLUE_Connectivity and BLUEInfluencers_Disparity are omitted from the regression model in the second attempt to solve the multicolinearity issue. The newest VIF values in the analysis in Table 4.13 below appear just fine.

Table 4.15. Regression Result for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT all Metrics (Multicolinearity Analysis-Step 3)

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.956[a]	.914	.904	.022	.914	91.785	3	28	.000

a. Predictors: (Constant), BLUE_Stability, BLUESensors_Disparity, BLUE_Eigenvalues

b. Dependent Variable: PBWin

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Lower Bound	Upper Bound	Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	-.638	.142		-4.478	.000	-.931	-.345					
	BLUE_Eigenvalues	.697	.068	.735	10.818	.000	.562	.831	.882	.901	.812	.693	1.443
	BLUESensors_Disparity	-.031	.006	-.324	-5.494	.000	-.043	-.019	-.111	-.733	-.316	.951	1.051
	BLUE_Stability	-.029	.006	-.353	-5.115	.000	-.040	-.017	-.693	-.708	-.295	.697	1.435

a. Dependent Variable: PBWin

When the experiment results of the model 8.5.8 with the perfectly newest VIF values calculated by applying eigenvalue, sensors disparity, and stability are examined in the Table 4.15 above, a regression analysis yields the same coefficient of determination, R square (R^2) value as 0.914 and provides the following equation:

$$y = 0.697x_1 - 0.031x_2 - 0.029x_3 - 0.638$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of a configuration

x_2: the sensors disparity value of a configuration

x_3: the stability value of a configuration

4.2.3.3. The Analysis of Each Experiment With Respect To Eigenvalue, Disparity, Stability

Table 4.16. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Stability

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	1	1			
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.942	.888	.804	.033	.023*
Model5.4.5	1	1			
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.966	.933	.920	.029	.000
Model6.4.6	.934	.873	.797	.038	.011**
Model6.5.6	1	1			
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.980	.960	.957	.024	.000
Model7.4.7	.959	.920	.910	.023	.000
Model7.5.7	.939	.882	.812	.028	.009**
Model7.6.7	1	1			
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.983	.965	.964	.023	.000
Model8.4.8	.962	.926	.923	.027	.000
Model8.5.8	.956	.914	.904	.022	.000
Model8.6.8	.954	.910	.0856	.036	.005**
Model8.7.8	1	1			
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.980	.960	.959	.027	.000
Model9.4.9	.978	.957	.957	.022	.000
Model9.5.9	.973	.947	.946	.020	.000
Model9.6.9	.967	.936	.928	.020	.000
Model9.7.9	.984	.967	.948	.015	.000
Model9.8.9	1	1			
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				
Model10.3.10	.980	.961	.960	.029	.000
Model10.4.10	.982	.964	.963	.022	.000
Model10.5.10	.972	.945	.945	.022	.000
Model10.6.10	.967	.935	.933	.021	.000
Model10.7.10	.955	.911	.901	.018	.000
Model10.8.10	.717	.513	.221	.034	.271
Model10.9.10	1	1			
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				

Table 4.16. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model11.3.11	.976	.953	.953	.034	.000
Model11.4.11	.983	.967	.967	.024	.000
Model11.5.11	.979	.958	.958	.022	.000
Model11.6.11	.975	.950	.950	.020	.000
Model11.7.11	.960	.921	.919	.021	.000
Model11.8.11	.879	.773	.748	.026	.000*
Model11.9.11	.636	.405	.048	.035	.419
Model11.10.11	1	1			
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.973	.947	.947	.038	.000
Model12.4.12	.981	.962	.962	.027	.000
Model12.5.12	.961	.924	.924	.032	.000
Model12.6.12	.974	.948	.948	.022	.000
Model12.7.12	.972	.944	.944	.020	.000
Model12.8.12	.970	.940	.939	.019	.000
Model12.9.12	.874	.764	.738	.026	.000**
Model12.10.12	.814	.662	.460	.038	.117
Model12.11.12	1	1			
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

- * The overall P value is low; but the individual P values are high. There is not a large enough dataset to have a perfect linear regression.
- ** The overall P values is low; but some of the individual P values are high. There is not a large enough dataset to have a perfect linear regression.

When the individual experiment results calculated by applying eigenvalues, sensors disparities, and coefficients are examined in Table 4.16 above, the results are almost identical with minor differences. The perfomance of each experiment drops as the ratio between Sensors/Influencers and Deciders drops. The experiments with less than about 30 iterations have insignificant results.

4.2.3.4. The Analysis of Decider Basis Experiments With Respect To Eigenvalue, Disparity, Coefficient

Table 4.17. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Coefficient

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.933	.871	.870	.054	.000
ModelX4X	.929	.863	.863	.048	.000
ModelX5X	.926	.857	.857	.042	.000
ModelX6X	.927	.860	.859	.034	.000
ModelX7X	.913	.833	.832	.033	.000
ModelX8X	.906	.822	.818	.031	.000
ModelX9X	.750	.563	.529	.035	.000*
ModelX10X	.516	.266	-.009	.079	.454
ModelX11X	1	1			
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by applying eigenvalues, Sensors disparities and stabilities along with the Deciders (column-wise) are examined in Table 4.17, above, the R and R square values of the first three columns are about 5% higher than the results calculated by just applying eigenvalues, total disparities, and robustnesses. The rest of the columns follow the same pattern.The experiments with 9 Deciders and further are also insignificant. The models with 9 Deciders looks significant as a whole model. But indeed, it is an insignificant model from individual independent variables perpective.

4.2.3.5. The Analysis of Sensor/Influencer Basis Experiments With Respect To Eigenvalue, Sensors Disparity, Stability

Table 4.18. Collective Regression Results for the Sensor/Influencer Basis Aggregated Data of each BLUE Combination vs. all RED Combinations WRT Eigenvalue, Disparity, and Stability

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	1	1			
Model5Y5	.909	.826	.751	.034	.005*
Model6Y6	.949	.900	.889	.031	.000
Model7Y7	.972	.945	.942	.023	.000
Model8Y8	.971	.943	.943	.025	.000
Model9Y9	.974	.950	.949	.025	.000
Model10Y10	.976	.953	.953	.024	.000
Model11Y11	.982	.964	.964	.023	.000
Model12Y12	.986	.973	.973	.020	.000

When the experiments results calculated by applying eigenvalues, powers, and stabilities along with the sensors/influencers (row-wise) are examined in Table 4.18, above, the R and R square values are about 1-2% higher than the results calculated by just applying eigenvalues, total disparities, and robustnesses. The row with five Sensors/Influencers seems significant; but its independent variables individually have higher P values due to insufficient number of iterations. This row is, in fact, insignificant.

4.3. PERFORMANCE OF EACH BLUE COMBINATION VS EACH RED COMBINATION

Each BLUE combination vs. each RED combination has a total number of 6,024,756 datasets. These datasets (iterations) are the sum of square of each experiment's total number of different meaningful arrangements. These datasets contain the probability of BLUE win (dependent variable) for each BLUE combination versus each RED combination, and metrics such as eigenvalue, disparity, robustness, power of deciders, strength of connectivity, and coefficient of deciders. In this section, the probability of BLUE win is studied for known combinations of each side. They were run in the SPSS Statistics 17.0 software package, as well.

4.3.1. THE ANALYSIS OF EXPERIMENTS WITH RESPECT TO EIGENVALUE

4.3.1.1. The Analysis of All Experiments With Respect To Eigenvalue

When Table 4.19 is examined, the eigenvalue is a fair predictor or performance metric by itself alone for a networked force. It must be enhanced by some other metrics to measure the performance or predict the probability of win of a networked force.

Table 4.19. Regression Result for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.815[a]	.664	.664	.112163106045	.664	5963827.899	2	6024753	.000

a. Predictors: (Constant), RED_Eigenvalues, BLUE_Eigenvalues

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Lower Bound	Upper Bound	Zero-order	Partial	Part	Toterance	VIF
1	(Constant)	.420	.001		566.248	.000	.419	.422					
	BLUE_Eigenvalues	.870	.000	.770	2954.509	.000	.869	.871	.453	.769	.697	.821	1.218
	RED_Eigenvalues	-.845	.000	-.748	-2870.831	.000	-.846	-.845	-.422	-.760	-.678	.821	1.218

a. Dependent Variable: PBWin

The multiple correlation coefficient, R, is 81.5% that indicates a fair relationship. The coefficient of determination, R Square (R^2), 66.4% of the variation in probability of BLUE win is explained by the model which is very moderate. The results of the linear regression yield the following equation:

$$y = 0.420 + 0.870x_1 - 0.845x_2$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of BLUE configuration

x_2: the λ_{PFE} value of RED configuration

4.3.1.2. The Analysis of Each Experiment With Respect To Eigenvalue

Table 4.20. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	.960	.922	.765	.050	.280
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.726	.527	.511	.092	.000**
Model5.4.5	1	1	1	.000	-
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.788	.620	.618	.095	.000**
Model6.4.6	.681	.464	.450	.101	.000**
Model6.5.6	.973	.947	.842	.017	.229
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.817	.667	.667	.103	.000
Model7.4.7	.713	.508	.507	.097	.000
Model7.5.7	.714	.509	.497	.086	.000**
Model7.6.7	.577	.333	-1.000	.067	.867
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.834	.695	.695	.103	.000
Model8.4.8	.772	.596	.596	.100	.000
Model8.5.8	.665	.442	.440	.097	.000
Model8.6.8	.734	.539	.527	.092	.000**
Model8.7.8	.686	.471	-.588	.100	.728
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.856	.734	.734	.107	.000
Model9.4.9	.802	.644	.644	.103	.000
Model9.5.9	.739	.547	.547	.099	.000
Model9.6.9	.687	.472	.471	.100	.000
Model9.7.9	.542	.294	.276	.095	.000**
Model9.8.9	.875	.766	.299	.083	.483
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				
Model10.3.10	.860	.740	.740	.112	.000
Model10.4.10	.817	.668	.668	.106	.000
Model10.5.10	.761	.579	.579	.102	.000
Model10.6.10	.720	.519	.519	.100	.000
Model10.7.10	.612	.374	.373	.090	.000
Model10.8.10	.586	.343	.326	.088	.000
Model10.9.10	.981	.963	.889	.033	.192
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				

Table 4.20. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model11.3.11	.868	.754	.754	.117	.000
Model11.4.11	.832	.693	.693	.111	.000
Model11.5.11	.786	.618	.618	.108	.000
Model11.6.11	.725	.525	.525	.105	.000
Model11.7.11	.687	.472	.472	.102	.000
Model11.8.11	.569	.324	.323	.097	.000
Model11.9.11	.491	.241	.221	.089	.000**
Model11.10.11	.944	.892	.675	.050	.329
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.862	.743	.743	.125	.000
Model12.4.12	.843	.710	.710	.113	.000
Model12.5.12	.790	.624	.624	.112	.000
Model12.6.12	.751	.563	.563	.107	.000
Model12.7.12	.700	.490	.490	.103	.000
Model12.8.12	.677	.458	.458	.099	.000
Model12.9.12	.554	.307	.305	.093	.000
Model12.10.12	.575	.331	.314	.093	.000**
Model12.11.12	1	1	1	.000	-
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

When the individual experiment results calculated by just the eigenvalues are examined in Table 4.20 above, the experiments with less than approximately 30 different meaningful combinations (iterations) have insignificant results.

The multiple correlation coefficients, Rs, are about 75% that indicates a fair relationship. The coefficients of determination, R Square (R^2), are about 55% of the variation in probability of BLUE win is explained by the model which is very moderate.

4.3.1.3. The Analysis of Decider Basis Experiments With Respect To Eigenvalue

Table 4.21. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.862	.744	.744	.120	.000
ModelX4X	.839	.704	.704	.112	.000
ModelX5X	.788	.621	.621	.111	.000
ModelX6X	.747	.558	.558	.107	.000
ModelX7X	.698	.488	.488	.103	.000
ModelX8X	.670	.448	.448	.099	.000
ModelX9X	.544	.296	.295	.092	.000
ModelX10X	.554	.307	.290	.094	.000
ModelX11X	.975	.951	.902	.023	.049*
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by just the eigenvalues along with the deciders (column-wise) are examined in Table 4.21, above, the R and R square values are low; they decrease gradually as the number of deciders increases. The experiments are significant until 11 deciders.

4.3.1.4. The Analysis of Sensor/Influencer Basis Experiments With Respect To Eigenvalue

Table 4.22. Collective Regression Results for the Sensor/Influencer Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	.931	.866	.733	.063	.134
Model5Y5	.728	.530	.516	.089	.000
Model6Y6	.770	.593	.591	.096	.000
Model7Y7	.795	.633	.632	.101	.000
Model8Y8	.804	.646	.646	.101	.000
Model9Y9	.819	.671	.671	.104	.000
Model10Y10	.817	.667	.667	.106	.000
Model11Y11	.819	.671	.671	.110	.000
Model12Y12	.814	.663	.663	.113	.000

When the experiments results calculated by applying eigenvalues along with the sensors/influencers (row-wise) are examined in Table 4.22, above, the R and R square values are stuck to lower 80% and mid-60%, respectively.

4.3.2. THE ANALYSIS OF EXPERIMENT WITH RESPECT TO EIGENVALUE, TOTAL DISPARITY, AND ROBUSTNESS

4.3.2.1. The Analysis of All Experiments With Respect To Eigenvalue, Total Disparity, and Robustness

Table 4.23. Regression Result for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Disparity, and Robustness

Descriptive Statistics

	Mean	Std. Deviation	N
PBWin	.47818620	.193616202	6024756
BLUE_Eigenvalues	2.34332659	.171281309	6024756
RED_Eigenvalues	2.34332659	.171281309	6024756
BLUE_Total_Disparity	7.74	2.335	6024756
RED_Total_Disparity	7.74	2.335	6024756
BLUE_Robustness	7.91	1.667	6024756
RED_Robustness	7.91	1.667	6024756

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.888[a]	.789	.789	.088920171	.789	3756576.700	6	6024749	.000

a. Predictors: (Constant), RED_Robustness, BLUE_Eigenvalues, BLUE_Total_Disparity, RED_Total_Disparity, BLUE_Robustness, RED_Eigenvalues

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Lower Bound	Upper Bound	Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	.422	.001		647.169	.000	.421	.423					
	BLUE_Eigenvalues	.607	.001	.537	1181.503	.000	.606	.608	.453	.434	.221	.170	5.895
	RED_Eigenvalues	-.580	.001	-.513	-1130.191	.000	-.581	-.579	-.422	-.418	-.211	.170	5.895
	BLUE_Total_Disparity	-.013	.000	-.156	-513.767	.000	-.013	-.013	-.178	-.205	-.096	.378	2.648
	RED_Total_Disparity	.013	.000	.162	530.694	.000	.013	.013	.204	.211	.099	.378	2.648
	BLUE_Robustness	.026	.000	.226	547.702	.000	.026	.026	.583	.218	.102	.206	4.847
	RED_Robustness	-.027	.000	-.235	-571.402	.000	-.027	-.027	-.582	-.227	-.107	.206	4.847

a. Dependent Variable: PBWin

When the experiments results of collected 6,024,756 datasets calculated by applying eigenvalues, disparities, and robustnesses are examined in Table 4.23 above, a regression analysis of the λ_{PFE}, the disparity and the robustness values yields an 18.8% increase in the coefficient of determination, the R square (R^2) forms a value of 0.664 to 0.789 and provides the following equation:

$$y = 0.422 + 0.607x_1 - 0.580x_2 - 0.013x_3 + 0.013x_4 + 0.026x_5 - 0.027x_6$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of a BLUE configuration

x_2: the λ_{PFE} value of a RED configuration

x_3: the disparity value of a BLUE configuration

x_4: the disparity value of a RED configuration

x_5: the robustness value of a BLUE configuration

x_6: the robustness value of a RED configuration

4.3.2.2. The Analysis of Each Experiment With Respect To Eigenvalue, Disparity, and Robustness

Table 4.24. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	.960	.922	.765	.050	.280
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.755	.571	.526	.091	.000*
Model5.4.5	1	1	1	.000	
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.831	.691	.685	.086	.000**
Model6.4.6	.713	.508	.468	.100	.000*
Model6.5.6	.973	.947	.842	.017	.229
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.870	.757	.756	.089	.000
Model7.4.7	.768	.590	.587	.088	.000**
Model7.5.7	.736	.542	.504	.085	.000*
Model7.6.7	.577	.333	-1.000	.067	.816
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.883	.779	.779	.088	.000
Model8.4.8	.819	.671	.670	.090	.000
Model8.5.8	.744	.554	.551	.087	.000**
Model8.6.8	.815	.664	.637	.081	.000*
Model8.7.8	.686	.471	-.588	.100	.728
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.908	.824	.824	.087	.000
Model9.4.9	.857	.735	.735	.089	.000
Model9.5.9	.804	.646	.646	.088	.000
Model9.6.9	.754	.568	.566	.091	.000**
Model9.7.9	.737	.544	.507	.078	.000*
Model9.8.9	.875	.766	.299	.083	.483
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				

Table 4.24. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model10.3.10	.918	.843	.843	.087	.000
Model10.4.10	.877	.769	.769	.088	.000
Model10.5.10	.827	.683	.683	.088	.000
Model10.6.10	.784	.614	.614	.090	.000
Model10.7.10	.682	.465	.462	.083	.000**
Model10.8.10	.640	.409	.361	.086	.000*
Model10.9.10	.981	.963	.889	.033	.192
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				
Model11.3.11	.928	.861	.861	.088	.000
Model11.4.11	.899	.809	.809	.088	.000
Model11.5.11	.861	.741	.741	.089	.000
Model11.6.11	.813	.662	.622	.089	.000
Model11.7.11	.773	.597	.597	.089	.000
Model11.8.11	.642	.412	.409	.091	.000**
Model11.9.11	.608	.369	.318	.083	.000*
Model11.10.11	.944	.892	.675	.050	.329
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.932	.868	.868	.089	.000
Model12.4.12	.909	.826	.826	.088	.000
Model12.5.12	.878	.770	.770	.088	.000
Model12.6.12	.837	.700	.700	.089	.000
Model12.7.12	.789	.622	.622	.089	.000
Model12.8.12	.745	.556	.555	.090	.000
Model12.9.12	.628	.394	.390	.087	.000**
Model12.10.12	.654	.427	.381	.088	.000*
Model12.11.12	1	1	1	.000	
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

When the individual experiment results calculated by eigenvalues, disparities, and robustnesses are examined in Table 4.24 above, the significant experiments have the multiple correlation coefficients, Rs, varying from 74.5% to 93.2% that indicate a good relationship. The coefficients of determination, R Square (R^2) - varying from 55.6% to 86.8%, of the variation in probability of BLUE win are explained by the model that are fair.

4.3.2.3. The Analysis of Column-wise Experiments With Respect To Eigenvalue, Disparity, and Robustness

Table 4.25. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.927	.860	.860	.089	.000
ModelX4X	.904	.818	.818	.088	.000
ModelX5X	.874	.763	.763	.088	.000
ModelX6X	.833	.694	.694	.089	.000
ModelX7X	.786	.619	.619	.089	.000
ModelX8X	.738	.544	.544	.090	.000
ModelX9X	.619	.383	.380	.087	.000**
ModelX10X	.606	.368	.320	.093	.000*
ModelX11X	1	1	1	.000	
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by eigenvalues, disparities, and robustnesses along with the deciders (column-wise) are examined in Table 4.25 above, the R values for significant experiments vary from 73.8% to 92.7%, and the R square values for significant experiments vary from 54.4% to 86%. They decrease exponentially as the number of deciders increases. The experiments are significant up to 9 Deciders.

4.3.2.4. The Analysis of Row-wise Experiments With Respect To Eigenvalue, Total Disparity, and Robustness

Table 4.26. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Disparity, and Robustness

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	.979	.959	.837	.050	.255
Model5Y5	.756	.571	.530	.087	.000*
Model6Y6	.811	.658	.653	.088	.000
Model7Y7	.848	.719	.718	.088	.000
Model8Y8	.852	.727	.727	.089	.000
Model9Y9	.873	.762	.762	.088	.000
Model10Y10	.877	.768	.768	.088	.000
Model11Y11	.888	.788	.788	.088	.000
Model12Y12	.889	.791	.791	.089	.000

When the experiments results calculated by applying eigenvalues, disparities, and robustnesses along with the Sensors/Influencers (row-wise) are examined in Table 4.26 above, the R values are increased logarithmically from 0.811 to 0.889 as the number of Sensors/Influencers increases, likewise the R square values are increased logarithmically from 0.658 to 0.791. They look like they are asymptotic to 0.9 and 0.8, respectively.

4.3.3. THE ANALYSIS OF ALL EXPERIMENTS WITH RESPECT TO ALL METRICS

Table 4.27. Regression Result for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT All Metrics

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.591[a]	.349	.349	.158166295051	.349	3236038.849	1	6024754	.000
2	.854[b]	.729	.729	.100862068398	.379	8418272.986	1	6024753	.000
3	.859[c]	.737	.737	.099215141985	.009	201677.434	1	6024752	.000
4	.879[d]	.772	.772	.092437890886	.035	915817.345	1	6024751	.000
5	.883[e]	.780	.780	.090792681422	.008	220322.335	1	6024750	.000
6	.888[f]	.789	.789	.088923690518	.009	255917.863	1	6024749	.000
7	.889[g]	.790	.790	.088740137758	.001	24950.319	1	6024748	.000
8	.891[h]	.795	.795	.087717097680	.005	141353.176	1	6024747	.000
9	.892[i]	.795	.795	.087710650839	.000	886.685	1	6024746	.000
10	.892[j]	.795	.795	.087707057592	.000	494.663	1	6024745	.000
11	.892[k]	.795	.795	.087599907205	.001	14748.700	1	6024744	.000
12	.892[l]	.795	.795	.087593995994	.000	814.178	1	6024743	.000
13	.892[m]	.795	.795	.087593681370	.000	47.031	1	6024742	.000

a. Predictors: (Constant), RED_Connectivity

b. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity

c. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues

d. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues

e. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity

f. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity

g. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength

h. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength

i. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength, RED_Stability

j. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength, RED_Stability, RED_Power

k. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength, RED_Stability, RED_Power, BLUE_Power

l. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength, RED_Stability, RED_Power, BLUE_Power, BLUE_Robustness

m. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, BLUE_Eigenvalues, RED_Eigenvalues, RED_Total_Disparity, BLUE_Total_Disparity, RED_Strength, BLUE_Strength, RED_Stability, RED_Power, BLUE_Power, BLUE_Robustness, BLUE_Stability

When the experiments results of collected 6,024,756 datasets calculated by applying all metrics are examined in Table 4.27 above, the result of best performance has R and R square values of 0.892 and 0.795, respectively. They are close to each other and can be accepted as high. Since there is a perfect

linear relationship among the independent variables, the estimates for the model can be computed in five different ways.

Table 4.28. Regression Result for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Power, and Connectivity

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.892[a]	.795	.795	.087644566852	.795	3896162.379	6	6024749	.000

a. Predictors: (Constant), RED_Connectivity, BLUE_Connectivity, RED_Eigenvalues, BLUE_Power, BLUE_Eigenvalues, RED_Power

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Lower Bound	Upper Bound	Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	.426	.001		546.292	.000	.424	.427					
	BLUE_Eigenvalues	.292	.000	.259	765.060	.000	.292	.293	.453	.298	.141	.298	3.360
	RED_Eigenvalues	-.260	.000	-.230	-680.122	.000	-.261	-.259	-.422	-.267	-.125	.298	3.360
	BLUE_Power	.098	.000	.455	788.326	.000	.098	.099	.486	.306	.145	.102	9.802
	RED_Power	-.101	.000	-.467	-809.020	.000	-.101	-.101	-.485	-.313	-.149	.102	9.802
	BLUE_Connectivity	-.006	.000	-.110	-244.592	.000	-.007	-.006	-.590	-.099	-.045	.168	5.948
	RED_Connectivity	.007	.000	.115	256.655	.000	.007	.007	.591	.104	.047	.168	5.948

a. Dependent Variable: PBWin

After the multicolinearity check, a regression analysis of the λ_{PFE}, the power, and the connectivity values yields very small increase in the R and R square values by 0.45% and 0.76%, respectively. This yields the following equation:

$$y = 0.426 + 0.292x_1 - 0.260x_2 + 0.098x_3 - 0.101x_4 - 0.006x_5 + 0.007x_6$$

Where, y: the average probability of a BLUE win for that configuration

x_1: the λ_{PFE} value of a BLUE configuration

x_2: the λ_{PFE} value of a RED configuration

x_3: the power value of a BLUE configuration

x_4: the power value of a RED configuration

x_5: the connectivity value of a BLUE configuration

x_6: the connectivity value of a RED configuration

4.3.3.1. The Analysis of Each Experiment With Respect To Eigenvalue, Power, and Connectivity

Table 4.29. Collective Regression Results for the Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Power, and Connectivity

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3.3.3	There is only one iteration that is not enough to calculate linear regression.				
Model4.3.4	.960	.922	.765	.050	.280
Model4.4.4	There is only one iteration that is not enough to calculate linear regression.				
Model5.3.5	.755	.570	.525	.091	.000*
Model5.4.5	1	1	1	.000	
Model5.5.5	There is only one iteration that is not enough to calculate linear regression.				
Model6.3.6	.836	.699	.694	.085	.000**
Model6.4.6	.711	.506	.466	.100	.000*
Model6.5.6	.973	.947	.842	.017	.229
Model6.6.6	There is only one iteration that is not enough to calculate linear regression.				
Model7.3.7	.872	.760	.759	.088	.000**
Model7.4.7	.766	.587	.584	.89	.000**
Model7.5.7	.737	.543	.506	.085	.000*
Model7.6.7	.577	.333	-1.000	.067	.816
Model7.7.7	There is only one iteration that is not enough to calculate linear regression.				
Model8.3.8	.885	.783	.783	.087	.000
Model8.4.8	.827	.684	.683	.089	.000
Model8.5.8	.743	.552	.549	.087	.000**
Model8.6.8	.813	.660	.633	.081	.000*
Model8.7.8	.686	.471	-.588	.100	.728
Model8.8.8	There is only one iteration that is not enough to calculate linear regression.				
Model9.3.9	.911	.830	.830	.086	.000
Model9.4.9	.860	.740	.740	.088	.000
Model9.5.9	.805	.649	.648	.088	.000
Model9.6.9	.748	.560	.557	.092	.000**
Model9.7.9	.722	.522	.483	.080	.000*
Model9.8.9	.875	.766	.299	.083	.483
Model9.9.9	There is only one iteration that is not enough to calculate linear regression.				
Model10.3.10	.920	.847	.847	.086	.000
Model10.4.10	.880	.775	.775	.087	.000
Model10.5.10	.830	.688	.688	.088	.000
Model10.6.10	.788	.621	.621	.089	.000
Model10.7.10	.679	.462	.458	.083	.000**
Model10.8.10	.642	.412	.365	.086	.000*
Model10.9.10	.981	.963	.889	.033	.192
Model10.10.10	There is only one iteration that is not enough to calculate linear regression.				

Table 4.29. (continued)

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model11.3.11	.930	.865	.865	.087	.000
Model11.4.11	.902	.814	.814	.086	.000
Model11.5.11	.864	.747	.747	.088	.000
Model11.6.11	.817	.667	.667	.088	.000
Model11.7.11	.779	.607	.607	.088	.000**
Model11.8.11	.640	.410	.407	.091	.000*
Model11.9.11	.604	.364	.313	.083	.000*
Model11.10.11	.944	.892	.675	.050	.329
Model11.11.11	There is only one iteration that is not enough to calculate linear regression.				
Model12.3.12	.935	.873	.873	.088	.000
Model12.4.12	.912	.831	.831	.086	.000
Model12.5.12	.881	.777	.777	.087	.000
Model12.6.12	.841	.708	.708	.088	.000
Model12.7.12	.792	.627	.627	.088	.000
Model12.8.12	.751	.564	.564	.089	.000**
Model12.9.12	.632	.399	.396	.086	.000**
Model12.10.12	.653	.426	.379	.088	.000*
Model12.11.12	1	1	1	.000	
Model12.12.12	There is only one iteration that is not enough to calculate linear regression.				

When the individual experiment results calculated by eigenvalues, powers, and connectivies are examined in Table 4.29 above, the significant experiments have the multiple correlation coefficients, Rs, varying from 78.8% to 93.5% that indicate a good relationship. The coefficients of determination, R Square (R^2) - varying from 62.1% to 87.3%, of the variation in probability of a BLUE win are explained by the model that are fair.

4.3.3.2. The Analysis of Decider Basis Experiments With Respect To Eigenvalue, Power, and Connectivity

Table 4.30. Collective Regression Results for the Decider Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Power, and Connectivity

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
ModelX3X	.930	.864	.864	.088	.000
ModelX4X	.907	.823	.823	.087	.000
ModelX5X	.877	.770	.770	.087	.000
ModelX6X	.837	.701	.701	.088	.000
ModelX7X	.789	.623	.623	.088	.000
ModelX8X	.744	.553	.553	.089	.000
ModelX9X	.624	.389	.386	.086	.000**
ModelX10X	.610	.372	.324	.093	.000*
ModelX11X	1	1	1	.000	
ModelX12X	There is only one iteration that is not enough to calculate linear regression.				

When the experiments results calculated by eigenvalues, powers, and connectivites along with the deciders (column-wise) are examined in Table 4.30 above, the R values for significant experiments vary from 74.4% to 93%, and the R square values for significant experiments vary from 55.3% to 86.4%. They decrease exponentially as the number of deciders increases. The experiments are significant up to 9 Deciders.

4.3.3.3. The Analysis of Row-wise Experiments With Respect To Eigenvalue, Total Disparity, and Robustness

Table 4.31. Collective Regression Results for the Sensor/Influencer Basis Aggregated Data of each BLUE Combination vs. each RED Combination WRT Eigenvalue, Power, and Connectivity

Model	R	R Square	Adjusted R Square	Std.Error of the Estimate	Sig.
Model3Y3	There is only one iteration that is not enough to calculate linear regression.				
Model4Y4	.979	.959	.837	.050	.255
Model5Y5	.757	.573	.531	.088	.000*
Model6Y6	.816	.666	.661	.087	.000**
Model7Y7	.849	.721	.720	.088	.000**
Model8Y8	.856	.733	.733	.088	.000
Model9Y9	.876	.767	.767	.088	.000
Model10Y10	.880	.774	.774	.087	.000
Model11Y11	.891	.793	.793	.087	.000
Model12Y12	.893	.798	.798	.088	.000

When the experiments results calculated by applying eigenvalues, disparities, and robustnesses along with the sensors/influencers (row-wise) are examined in Table 4.31 above, the R values are increased logarithmically from 0.856 to 0.893 as the number of Sensors/Influencers increases, likewise the R square values are increased logarithmically from 0.733 to 0.798. They look like they are asymptotic to 0.9 and 0.8, respectively.

5. CONCLUSION

5.1. GENERAL EVALUATION OF THE RESEARCH PURPOSE

The purpose of this research is to explore what causes Network Centric Operations to be effective and the influence of network factors on NCOs.

This research is the second attempt to identify up to what configuration the utility of the Perron-Frobenius Eigenvalue (λ_{PFE}) can be determined as a good metric to predict the perfomance of a network in general and particularly combat power of the Information Age (Cares, 2005).

The only known parameter about each experiment is a specially designed binary coded adjacency matrix according to the defined rules in Table 2.1. The adjacency matrix points out the relationships between the entites. Each entity is initially displaced randomly. Then entities except for Deciders move around according to rule set forth to do their designated functions: sense, track, shoot, kill, and move. From only that adjacency matrix in hand, that differs solely in entities arrangements, various metrics have been derived to measure the ability of a network to generate the feedback effects in general and combat power in the environment of the Information Age Combat Model (Cares, 2005). The total of 55 experiments with various force combinations were executed to test its effectiveness and influence in an agent based simulation model.

The Sensor-Decider-Influencer triad as a squad (minimum structure) of a war unit is interdependent to sensors and influencers since Deciders are accepted as everlasting entities during the experiment. The war unit without Sensors can not sense and track; likewise it can not shoot and kill without Influencers, either. The war unit with equal number of Sensors and Influencers (called as balanced) is more effective and durable for the war job. The war unit without the other half is not effective; it is no longer a war unit in the battlefield, it just waits to be killed.

When the probability of a BLUE win is ranked from lowest to highest for its each combination vs. all RED combinations, the BLUE force with maximum unbalanced (completely scattered deciders) has the lowest probability of BLUE win; the BLUE force with maximum balanced deciders has the highest probability of BLUE win. Intermediate values lay between these two extreme combinations. For example, the war unit that one of its Deciders with one Sensor and maximum Influencers, the other Decider with maximum Sensors and one Influencer and the rest of its Deciders with one Sensor and one Influencer can be thought as a *maximum unbalanced* war unit. A war unit that has one of its Deciders with maximum Sensors and Influencers and the rest of its Deciders with one Sensor and one Influencer can be thought as a *maximum balanced* war unit. The mid-points in the ranking are almost evenly balanced (have almost the same number of Sensors and Influencers) war unit (i.e., each Decider has two Sensors and two Influencers or three Sensors and three Influencers, etc.; a minor deviation might have seen due to randomness). The more balanced the war unit, the better the performance of a networked force.

The eigenvalues, disparity, robustness, strength, power, stability, and connectivity are some metrics generated from the different meaningful combinations of Sensors and Influencers linked to each Decider by applying various operations described earlier. These metrics are the tools to detect the maximum points from unbalanced to balanced intervals. Some of the metrics are Integers and some of them are real numbers to give the balance issue a weight; low number if it is unbalanced, there is high number otherwise.

The results of 55 experiments with each BLUE combination vs all RED combinations in the agent-based simulation modeling presented in this research show that the multiple correlation coefficient, R, is 58% and the coefficients of determination, R Square (R^2) is 33.6%. There is a very poor degree of correlation between the λ_{PFE} value and the average probability of a BLUE win. Therefore, the value of the λ_{PFE} is a very poor metric by itself to measure the performance of an Information Age Combat Force.

The results of 55 experiments with each BLUE combination vs each RED combination in the agent-based simulation modeling presented in this research show that the multiple correlation coefficient, R, is 81.5% and the coefficients of determination, R Square (R^2) is 66.4%. There is a very fair degree of positive correlation between the BLUE λ_{PFE} value and the average probability of a BLUE win; there is a very fair degree of negative corelation between the RED λ_{PFE} value and the average probability of a BLUE win. Therefore, the value of the λ_{PFE} is a very fair predictor or metric by itself to measure the performance of an Information Age Combat Force.

While the λ_{PFE} value alone was a sufficient predictor as poor/fair for a networked forces up to with seven (excluded) Deciders, it was not a sufficient predictor by itself for a networked force with larger than or equal to seven Deciders. As the ratio between the the number of distinct eigenvalues and the number of different meaningful combinations decreases as the number of Sensor-Influencer and Decider increases. This effect diminishes the power of the λ_{PFE} value as a metric to measure the probability of a BLUE win. So additional metrics should be taken into consideration to measure the performance of a networked force.

Two additional metrics introduced before (Deller, 2009) are applied with the eigenvalues increasing the performance measure of a networked force very significantly in the results of 55 experiments with each BLUE combination vs all RED combinations. The multiple correlation coefficient, R, is increased from 0.580 to 0.891 by 53.62%; and the coefficients of determination, R square, is increased from 0.336 to 0.794 by 136.3%. There are a fair degree of positive corelation (0.580) in between the eigenvalue and the probability of a BLUE win, a poor degree of negative corelation (-0.232) in between the disparity and the probability of a BLUE win and a good degree of positive corelation (0.838) in between the robustness and the probability of a BLUE win.

When the same metrics are applied to the results of 55 experiments with each BLUE combination vs each RED combination in the agent-based simulation modeling, they increase the the performance measure of a networked force well.

The multiple correlation coefficient, R, is increased from 0.815 to 0.888 by 8.95%; and the coefficients of determination, R square, is increased from 0.664 to 0.789 by 18.83%. There are a fair degree of positive corelations (0.483) between the BLUE eigenvalue and the probability of a BLUE win, a poor degree of negative corelation (-0.178) between the BLUE total disparity and the probability of a BLUE win and a fair degree of positive corelation (0.583) in between the BLUE robustness and the probability of a BLUE win, a fair degree of negative corelation (-0.422) between the RED eigenvalue and the probability of a BLUE win, a poor degree of positive corelation (0.204) between the RED total disparity and the probability of a BLUE win and a fair degree of negative corelation (-0.582) between the BLUE robustness and the probability of a BLUE win.

The additional new metrics, *power and connectivity*, introduced in this research can increase the performance measure of a networked force better once they are applied together with the previous metrics to the results of 55 experiments with each BLUE combination vs all RED combinations. The R value is increased from 0.891 to 0.983 by 10.32% and the R square valeu is increased from 0.794 to 0.966 by 21.66%. There are a fair positive degrees of corelation (0.528) between the power and the probability of a BLUE win, a good degree of negative corelation (-0.866) between the connectivity and the probability of a BLUE win.

When the eigenvalue, the power and the connectivity values are applied to the results of 55 experiments with each BLUE combination vs each RED combination as metrics to measure the performance of a networked force, they yield a little bit better performance, less than 1%. The R value is slightly increased from 0.888 to 0.892 by 0.45%, and the R square value is also slightly

increased from 0.789 to 0.795 by 0.76%. There are a fair degree of positive corelations (0.453) between the BLUE eigenvalue and the probability of a BLUE win, a fair degree of positive corelation (0.486) between the BLUE power and the probability of a BLUE win, a fair degree of negative corelation (-0.590) between the BLUE connectivity and the probability of a BLUE win, a fair degree of negative corelation (-0.422) between the RED eigenvalue and the probability of a BLUE win, a poor degree of negative corelation (-0.485) between the RED power and the probability of a BLUE win and a fair degree of positive corelation (0.591) between the RED connectivity and the probability of a BLUE win.

5.2. RECOMMENDATIONS FOR FUTURE WORK

There is still plenty of room to explore in the agent-based modeling of the Information Age Combat Modeling.

Java code based on the mathematical function defined in this research runs fast to a certain point then it turns out to be cumbersome script that looks like it is frozen. It runs fast for a small number of Deciders, but when the number of Deciders is increased, the computation time gets higher exponentially. The code ran for almost a month for 30 Sensors, 6 Deciders, and 30 Influencers; but it could not finish running the code in cluster of lunix High Performance Computer Group. The same mathematical function or a better one can be created and converted into a better performing environment.

A more powerful agent-based modeling and simulation environment supporting 64-bit operating system can be used to explore a larger research space. A 32-bit opearing system has a memory issue, it can allocate up to 3 GB RAM memory. If a large model is run, there are two options; either split the inputs into small groups and run them individually, then gather the data (it takes a lot of time) or run the whole model in 64-bit operating system in cluster. A 64-bit operating system has enough memory allocation to run larger models. NetLogo and AnyLogic are both java based agent-based modeling and simulation packages. NetLogo does not need user to know Java to build the models in it;

but it is very cumbersome and not flexible to varying stuations. AnyLogic is very powerful and flexible, but it is does not support a 64-bit operating system, and user needs to know Java coding to build the models.

A significant contribution will be to add some links and define some functions accordingly to activate the inactive deciders that have neither a Sensor nor an Influencer. "Echelon" links between Sensors to Sensors, Deciders to Deciders, and Influencers to Influencers, such as link types 1, 5, 11, or direct coordination links from Sensors to Influencers, such as a link type 9, should be thought as a good contribution for the future work. Moreover, Deciders are set forth as invulnerable targets for opposing Influencers. Without a Sensor or an Influencer, Deciders are set aside. Letting the Deciders be vulnerable Targets for opposing Influencers make the models more realistic combined with the proposed links for the future work. These additional links and rules will definitely increase the performance of a networked force and change its adjacency matrix structure.

Multiple regression analysis with the interactions of the metrics will be a good research area for the future work.

Both Sensors and Influencers with identical features are used in this research. Different research for the whole search space will be a good study for varying sensing and influencing ranges.

It is also a good contribution to analyze the performance of networked forces of unequal assets.

The whole experiments are done deterministically. The biggest contribution could be to redesign and analyse the whole model with new rules in a stochastic manner.

5.3. SUMMARY

The concept of attack, defense, and security in the twenty-first century is very robust and dynamic as the threat changes in the Information Age. There is

no pitched battle anymore that require large units from both sides. There are regional or local battles that require small units that are used more effectively. For security reasons, geographically dispersed and functionally diverse units are required. The challenge is how to orchestrate or control these units for their intended purpose. How does command and control function? What type of units are required? The answers to these questions are obviously a complex matter. The concept of distributed networked operations must be understood thoroughly in order to command and control the required units effectively. The entities represent the units and the links represent the relationship in between them. If some quantifiable metrics (parameters) that represent the characteristics of the distributed networked operations are comprehended, then it is easy to construct the units for the intended purpose and orchestrate them accordingly. There is not just a good quantifiable metric that can explain the relationship between the nodes, in general, as the network structure grows. But the combination of the metrics that are derived from a nodes partitioning structure can explain the relationship more precisely. The structure of the networked centric operations as in the Information Age Combat Modeling is also applicable for non-military applications for distributed, networked operations.

REFERENCES

1. Deller, Sean. Towards The Quantitative Analysis Of The Connectivity Value Of Networked Operations, Ph.D.Dissertation, Old Dominion University, Norfolk, VA, USA, 2009.

2. Cares, Jeff. Distributed Networked Operations. iUniverse, New York, 2005.

3. Honabarger, Jason B., Modeling Network Centric Warfare (NCW) With The System Effectiveness Analysis Simulation (SEAS),M.S.Thessis, Air Force Institute of Technology, Dayton, OH, USA, 2006.

4. Tolk, Andreas, Bowen, Robert J., and Hester, Patrick T. Using Agent Technology to Move From Intention-Based to effect-Based Models. Winter Simulation Conference Proceedings, IEEE Press, 863-871, 2008.

5. Hanratty, Timothy, Dumer, John, Yen, John, Fan Xiaocong 2003. Using Agents with Shared Mental Model to Support Network-Centric Warfare. In Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), Orlando, USA, July 27-30, 2003.

6. David Alberts, John Garstka, Frederick Stein, Network Centric Warfare, CCRP, July 2002.

7. Wong-Jiru, Ann, Colombi, John, Suzuki, Laura, Robert Mills, Graph Theoretical Analysis of network Centric Operations Using Multi-Layer Models, Air Force Institute of Technology, Sep 2006.

8. Honda, Tomonari, Sato, Hirosji, Namatame, Akira, Evolutionary Learning in Agent-Based Combat Simulation, Dept. of Computer Science, National Defense Academy, Japan, 2006.

9. Qing, Xue, Guo-Hui Zhang, Bing-Bing Lei, Xing-Dong Peng, The Study of C4ISR System Effectiveness Evaluation Based on Information Entropy in the Network Centric Warfare, International Conference on Intelligent Human-Machine Systems and Cybernetics, 2009.

10. McCormick, John M., Gerken, Peter M., Barry, Kevin P., Sidharta Brian, Achieving Battlespace Awareness in Network-Centric Warfare by Integrating Web and Agent Technologies, Lockheed Martin Advanced Technology Laboratories, 2004.

11. http://en.wikipedia.org/wiki/Network-centric warfare.

12. Lalbakhsh, Pooia, Sohrabi, Nasrin, Fesharaki, Mehdi N., The Role of Service Oriented Architecture in Battlefield Situational Awareness, Islamic AZAD University-Science & Research Campus, Tehran, IRAN, 2009.

13. http://en.wikipedia.org/wiki/Eigenvalue,_eigenvector_and_eigenspace.

14. R. L. Ott, An Introduction To Statistical Methods and Data Analysis, Fourth ed. Belmont, California: Duxbury Press, 1993.

15. A. Alin, "Multicolinearity," Wiley Interdisciplinary Reviews: Computational Statistics, vol. 2, pp. 370-374, 2010.

16. F. Ozturk and F. Akdeniz, "Ill-Conditioning and Multicolinearity," Linear Algebra and Its Applications, vol. 321, pp. 295-305, 2000.

17. G. Chartrand, Introductory Graph Theory. Mineola, New York: Dover Publications, 1985.

18. David S. Alberts, Richard E. Hayes, David A. Signori, Understanding Information Age Warfare. Washington D.C.: CCRP Press, August 2001.

APPENDIX A: JAVA CODES DIFFERENT MEANINGFUL COMBINATIONS

A.1. JAVA CODE FOR INTEGER PARTITIONING AND PERMUTATIONS ALGORITHM

```
import java.util.Vector;
public class AlgorithmLib {
        /**
        * This function returns a list containing all possible permutations of
        * the integer partition for the number n of size i. This is a recursive algorithm.
        * For brevity of explanation, suppose the name of the function intPartitionPerms is f.
        * Then f can be computed recursively as follows:
        *
        * 1. f(n, 1) = {{n}}     (base case)
        *
        * 2. f(n, i) = {{k} U f(n-k, i-1) | k = n-1...1 and n-k >= i-1}  (recursive step)
        *
        * @param n: The number to get integer partitions for
        * @param i: The size of the partition
```

```
 * @return  a Vector of Vectors if Integers
 */
public static Vector<Vector<Integer>> intPartionPerms(int n, int i) {
        // Initialize the list to hold the permutations
        Vector<Vector<Integer>> list = new Vector<Vector<Integer>>();
        // If i = 1, then the integer partition is just {{n}} - base case
        if(i == 1) {
                Vector<Integer> v = new Vector<Integer>();
                v.add(new Integer(n));
                list.add(v);
        }
        else {
                for(int k = n - 1; k >= 1; k --) {
                        // In cases where n-k > i-1, no solution exists - so ignore these
                        if((n - k) >= i - 1) {
                                // Recursive step for next k
                                Vector<Vector<Integer>> next = intPartionPerms(n - k, i - 1);
                                /* next is a set of permutations each of size i - 1. Here
```

```
                    we add {n} to the front of each permutation, turning
                    next into a set of permutations of length n. */
                  for(int j = 0; j < next.size(); j ++) {
                        Vector<Integer> curr = next.get(j);
                        curr.insertElementAt(new Integer(k), 0);
                  }
                  // Add this next set of permutations to the list.
                  list.addAll(next);
               }
            }
         }
         return list;
      }
}
```

A.2. JAVA CODE FOR INTEGER PARTITIONING AND PERMUTATIONS

```
import java.util.Vector;
public class IntegerPartitionPerms {
        // Read in n and i, and call the function.
        public static void main(String[] args) {
                int n = Integer.parseInt(args[0]);
                int i = Integer.parseInt(args[1]);
                System.out.println("");
                System.out.println("n = " + args[0] + ", i = " + args[1]);
                System.out.println("");
                System.out.println("All Integer Permutation Permutations:");
                print(AlgorithmLib.intPartionPerms(n, i));
                //print(new IntegerPartitionAlgorithm().solve(n, i));
        }

        // Provide a function for printing.
        public static void print(Vector<Vector<Integer>> lists) {
```

```
            for(int i = 0; i < lists.size(); i ++) {
                for(int j = 0; j < lists.get(i).size(); j ++) {
                    System.out.print(lists.get(i).get(j).toString() + "  ");
                }
                System.out.println("");
            }
        }
}
```

A.3. JAVA CODE FOR CROSSING ALGORITHM

```
import java.util.Hashtable;
import java.util.Vector;
import java.util.Collection;
import java.util.Collections;
import java.math.*;
public class CrossingAlgorithm {
        public static double tolerance = 0.0000000001;
        public static void main(String[] args) {
                int n1 = Integer.parseInt(args[0]);
                int i = Integer.parseInt(args[1]);
                int n2 = Integer.parseInt(args[2]);
                System.out.println("Crossing (n1, i, n2): " + args[0] + "," + args[1] + "," + args[2]);
                System.out.println("");
                // The line of code below illustrates how to call this.
                Vector<String> strings = new CrossingAlgorithm().crossToStrings(n1, i, n2);
                for(String combination : strings) {
                        System.out.println(combination);
```

```
        }
        System.out.println("Total Combinations: " + strings.size());
}
public Vector<Vector<Integer>> cross(int n1, int i, int n2) {
        Vector<Vector<Integer>> left = AlgorithmLib.intPartionPerms(n1, i);
        Vector<Vector<Integer>> top = AlgorithmLib.intPartionPerms(n2, i);
        Vector<Double> keys = new Vector<Double>();
        Vector<Vector<Integer>> combinations = new Vector<Vector<Integer>>();
        for(Vector<Integer> p1 : left) {
                for(Vector<Integer> p2 : top) {
                        int power = i;
                        double key = 0;
                        for(int j = 0; j < p1.size(); j ++) {
                                double a = Math.pow(p1.get(j).doubleValue(), ((int)power));
                                double b = Math.pow(p2.get(j).doubleValue(), (1/((double)power)));
                                key += a / b;
                        }
                        Vector<Integer> pair = new Vector<Integer>();
```

```
                    pair.addAll(p1);
                    pair.addAll(p2);
                    if(!containsKey(keys, key)) {
                        keys.add(new Double(key));
                        combinations.add(pair);
                    }
                }
            }
            return combinations;
        }
        private boolean containsKey(Vector<Double> keys, double key) {
            for(Double d : keys) {
                if(Math.abs(d.doubleValue() - key) <= tolerance)
                    return true;
            }
            return false;
        }
        public Vector<String> crossToStrings(int n1, int i, int n2) {
```

```
StringBuffer s = new StringBuffer();
Collection<Vector<Integer>> vals = cross(n1, i, n2);
Vector<String> strings = new Vector<String>();
for(Vector<Integer> pairs : vals) {
        s.append("{");
        boolean isFirst = true;
        for(Integer num : pairs) {
                if(!isFirst) {
                        s.append(",");
                }
                s.append(num.toString());
                isFirst = false;
        }
        s.append("}");
        strings.add(s.toString());
        s = new StringBuffer();
}
return strings;
```

```
        }
}
```

********** This is the end of Java Code **********

In command prompt, type:

Cross <number of sensors> <number of deciders> <number of influencers>

Like

cross 7 3 7

APPENDIX B: JAVA CODES OF IACM IN ANYLOGIC

```
package sidtmodel;
import java.sql.Connection;
import java.sql.SQLException;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Currency;
import java.util.Date;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Locale;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.SortedMap;
import java.util.SortedSet;
import java.util.Stack;
import java.util.Timer;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.Vector;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;

import static java.lang.Math.*;
import static com.xj.anylogic.engine.presentation.UtilitiesColor.*;
import static com.xj.anylogic.engine.presentation.UtilitiesDrawing.*;
import static com.xj.anylogic.engine.HyperArray.*;

import com.xj.anylogic.engine.*;
import com.xj.anylogic.engine.analysis.*;
import com.xj.anylogic.engine.connectivity.*;
import com.xj.anylogic.engine.connectivity.ResultSet;
import com.xj.anylogic.engine.connectivity.Statement;
import com.xj.anylogic.engine.presentation.*;

import java.awt.geom.Arc2D;

public class Main extends ActiveObject
{
```

```
  // Parameters

  public
int  nBDeciders;

  /**
   * Returns default value for parameter <code>nBDeciders</code>.
   */
  public
int  _nBDeciders_DefaultValue_xjal() {
    return
4
;
  }

  public void set_nBDeciders(
int  nBDeciders ) {
    if (nBDeciders == this.nBDeciders) {
      return;
    }
    this.nBDeciders = nBDeciders;
    onChange_nBDeciders();
    onChange();
  }

  void onChange_nBDeciders() {
      int index;
    index = 0;
    for ( Turtle object : influencersB ) {
      object.set_nFleets(nBDeciders);
      index++;
    }
    index = 0;
    for ( Turtle object : sensorsB ) {
      object.set_nFleets(nBDeciders);
      index++;
    }

  }


  public
double  sRange;

  /**
   * Returns default value for parameter <code>sRange</code>.
   */
  public
double  _sRange_DefaultValue_xjal() {
    return
10
;
  }

  public void set_sRange(
double  sRange ) {
    if (sRange == this.sRange) {
```

```
      return;
    }
    this.sRange = sRange;
    onChange_sRange();
    onChange();
  }

  void onChange_sRange() {
  }


  public
double  iRange;

  /**
   * Returns default value for parameter <code>iRange</code>.
   */
  public
double  _iRange_DefaultValue_xjal() {
    return
10
;
  }

  public void set_iRange(
double  iRange ) {
    if (iRange == this.iRange) {
      return;
    }
    this.iRange = iRange;
    onChange_iRange();
    onChange();
  }

  void onChange_iRange() {
  }


  public
int  BID;

  /**
   * Returns default value for parameter <code>BID</code>.
   */
  public
int  _BID_DefaultValue_xjal() {
    return 0;
  }

  public void set_BID(
int  BID ) {
    if (BID == this.BID) {
      return;
    }
    this.BID = BID;
    onChange_BID();
    onChange();
```

```
  }

  void onChange_BID() {
  }


  public
int  RID;

  /**
   * Returns default value for parameter <code>RID</code>.
   */
  public
int  _RID_DefaultValue_xjal() {
    return 0;
  }

  public void set_RID(
int  RID ) {
    if (RID == this.RID) {
      return;
    }
    this.RID = RID;
    onChange_RID();
    onChange();
  }

  void onChange_RID() {
  }


  public
int  seed;

  /**
   * Returns default value for parameter <code>seed</code>.
   */
  public
int  _seed_DefaultValue_xjal() {
    return 0;
  }

  public void set_seed(
int  seed ) {
    if (seed == this.seed) {
      return;
    }
    this.seed = seed;
    onChange_seed();
    onChange();
  }

  void onChange_seed() {
  }


  public
```

```
int  nRDeciders;

  /**
   * Returns default value for parameter <code>nRDeciders</code>.
   */
  public
int  _nRDeciders_DefaultValue_xjal() {
    return
4
;
  }

  public void set_nRDeciders(
int  nRDeciders ) {
    if (nRDeciders == this.nRDeciders) {
      return;
    }
    this.nRDeciders = nRDeciders;
    onChange_nRDeciders();
    onChange();
  }

  void onChange_nRDeciders() {
      int index;
    index = 0;
    for ( Turtle object : influencersR ) {
      object.set_nFleets(nRDeciders);
      index++;
    }
    index = 0;
    for ( Turtle object : sensorsR ) {
      object.set_nFleets(nRDeciders);
      index++;
    }

  }

  // Plain Variables

  public
int
 tick;
  public
Object[]
 result;
  public
int
 bWin;
  public
int
 rWin;
  // Events

  public EventTimeout event = new EventTimeout(this);


  @Override
```

```
  public String getNameOf( EventTimeout _e ) {
     if( _e == event ) return "event";
     return super.getNameOf( _e );
  }

  @Override
  public int getModeOf( EventTimeout _e ) {
    if ( _e == event ) return EVENT_TIMEOUT_MODE_CYCLIC;
    return super.getModeOf( _e );
  }

  @Override
  public double getFirstOccurrenceTime( EventTimeout _e ) {
    if ( _e == event ) return
0
;
    return super.getFirstOccurrenceTime( _e );
  }

  @Override
  public double evaluateTimeoutOf( EventTimeout _e ) {
    if( _e == event) return
1
;
    return super.evaluateTimeoutOf( _e );
  }

  @Override
  public void executeActionOf( EventTimeout _e ) {
    if ( _e == event ) {

tick++;
if (influencersR.size() + sensorsR.size() == 0) {
      // Blue team wins
      bWin = 1;
      event.reset();

//    getEngine().stop();
}

if (influencersB.size() + sensorsB.size() == 0) {
      // Red team wins
      rWin = 1;
      event.reset();
//    getEngine().stop();
}

reset();
sense();
track();
shoot();
kill();
moveInfluencers();
moveInfluencers();
moveInfluencers();
moveInfluencers();
moveInfluencers();
```

```
moveSensors();
moveSensors();
moveSensors();
moveSensors();
moveSensors();

;
      return ;
    }
    super.executeActionOf( _e );
  }
  // Embedded Objects


  public String getNameOf( ActiveObject ao ) {
    return null;
  }

  public ActiveObjectArrayList<Turtle> deciderB = new
ActiveObjectArrayList<Turtle>();
  public ActiveObjectArrayList<Turtle> deciderR = new
ActiveObjectArrayList<Turtle>();
  public ActiveObjectArrayList<Turtle> influencersB = new
ActiveObjectArrayList<Turtle>();
  public ActiveObjectArrayList<Turtle> influencersR = new
ActiveObjectArrayList<Turtle>();
  public ActiveObjectArrayList<Turtle> sensorsB = new
ActiveObjectArrayList<Turtle>();
  public ActiveObjectArrayList<Turtle> sensorsR = new
ActiveObjectArrayList<Turtle>();

  public String getNameOf( ActiveObjectCollection<?> aolist ) {
    if( aolist == deciderB ) return "deciderB";
    if( aolist == deciderR ) return "deciderR";
    if( aolist == influencersB ) return "influencersB";
    if( aolist == influencersR ) return "influencersR";
    if( aolist == sensorsB ) return "sensorsB";
    if( aolist == sensorsR ) return "sensorsR";
    return null;
  }

  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection deciderB<br>
   * @return newly created embedded object
   */
  public Turtle add_deciderB() {
    int index = deciderB.size();
    Turtle object = instantiate_deciderB_xjal( index );
    setupParameters_deciderB_xjal( object, index );
    create_deciderB_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection deciderB<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>deciderB.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_deciderB( int type, int nFleets, Color teamColor )
{
    int index = deciderB.size();
    Turtle object = instantiate_deciderB_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_deciderB_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection deciderB<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object deciderB - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to deciderB
   */
  public boolean remove_deciderB( Turtle object ) {
    if( ! deciderB._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection deciderR<br>
   * @return newly created embedded object
   */
  public Turtle add_deciderR() {
    int index = deciderR.size();
    Turtle object = instantiate_deciderR_xjal( index );
    setupParameters_deciderR_xjal( object, index );
    create_deciderR_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection deciderR<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>deciderR.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_deciderR( int type, int nFleets, Color teamColor )
{
    int index = deciderR.size();
    Turtle object = instantiate_deciderR_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_deciderR_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection deciderR<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object deciderR - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to deciderR
   */
  public boolean remove_deciderR( Turtle object ) {
    if( ! deciderR._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection influencersB<br>
   * @return newly created embedded object
   */
  public Turtle add_influencersB() {
    int index = influencersB.size();
    Turtle object = instantiate_influencersB_xjal( index );
    setupParameters_influencersB_xjal( object, index );
    create_influencersB_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection influencersB<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>influencersB.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_influencersB( int type, int nFleets, Color
teamColor ) {
    int index = influencersB.size();
    Turtle object = instantiate_influencersB_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_influencersB_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection influencersB<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object influencersB - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to influencersB
   */
  public boolean remove_influencersB( Turtle object ) {
    if( ! influencersB._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection influencersR<br>
   * @return newly created embedded object
   */
  public Turtle add_influencersR() {
    int index = influencersR.size();
    Turtle object = instantiate_influencersR_xjal( index );
    setupParameters_influencersR_xjal( object, index );
    create_influencersR_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection influencersR<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>influencersR.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_influencersR( int type, int nFleets, Color
teamColor ) {
    int index = influencersR.size();
    Turtle object = instantiate_influencersR_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_influencersR_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection influencersR<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object influencersR - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to influencersR
   */
  public boolean remove_influencersR( Turtle object ) {
    if( ! influencersR._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection sensorsB<br>
   * @return newly created embedded object
   */
  public Turtle add_sensorsB() {
    int index = sensorsB.size();
    Turtle object = instantiate_sensorsB_xjal( index );
    setupParameters_sensorsB_xjal( object, index );
    create_sensorsB_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection sensorsB<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>sensorsB.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_sensorsB( int type, int nFleets, Color teamColor )
{
    int index = sensorsB.size();
    Turtle object = instantiate_sensorsB_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_sensorsB_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection sensorsB<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object sensorsB - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to sensorsB
   */
  public boolean remove_sensorsB( Turtle object ) {
    if( ! sensorsB._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection sensorsR<br>
   * @return newly created embedded object
   */
  public Turtle add_sensorsR() {
    int index = sensorsR.size();
    Turtle object = instantiate_sensorsR_xjal( index );
    setupParameters_sensorsR_xjal( object, index );
    create_sensorsR_xjal( object, index );
    object.start();
    return object;
  }

  /**
```

```
   * This method creates and adds new embedded object in the replicated
embedded object collection sensorsR<br>
   * This method uses given parameter values to setup created embedded
object<br>
   * Index of this new embedded object instance can be obtained through
calling <code>sensorsR.size()</code> method <strong>before</strong>
this method is called
   * @param type
   * @param nFleets
   * @param teamColor
   * @return newly created embedded object
   */
  public Turtle add_sensorsR( int type, int nFleets, Color teamColor )
{
    int index = sensorsR.size();
    Turtle object = instantiate_sensorsR_xjal( index );
    // Setup parameters
    object.type = type;
    object.nFleets = nFleets;
    object.teamColor = teamColor;
    // Finish embedded object creation
    create_sensorsR_xjal( object, index );
    object.start();
    return object;
  }

  /**
   * This method removes the given embedded object from the replicated
embedded object collection sensorsR<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded
object sensorsR - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to sensorsR
   */
  public boolean remove_sensorsR( Turtle object ) {
    if( ! sensorsR._remove( object ) ){
      return false;
    }
    object.setDestroyed();
    return true;
  }

  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_deciderB_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, deciderB );

    deciderB._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
```

```
   */
  private void setupParameters_deciderB_xjal(Turtle object, final int
index ) {
    object.type =
3
;
    object.nFleets = object._nFleets_DefaultValue_xjal();
    object.teamColor =
lightSteelBlue
;
  }

  /**
   * Setups an embedded object instance<br>
   */
  private void create_deciderB_xjal(Turtle object, final int index ) {
    object.setEnvironment(
environment
);
    object.create();

    // Port connections
  }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_deciderR_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, deciderR );

    deciderR._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
   */
  private void setupParameters_deciderR_xjal(Turtle object, final int
index ) {
    object.type =
3
;
    object.nFleets = object._nFleets_DefaultValue_xjal();
    object.teamColor =
red
;
  }

  /**
   * Setups an embedded object instance<br>
   */
  private void create_deciderR_xjal(Turtle object, final int index ) {
    object.setEnvironment(
environment
);
    object.create();
```

```
    // Port connections
  }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_influencersB_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, influencersB );

    influencersB._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
   */
  private void setupParameters_influencersB_xjal(Turtle object, final
int index ) {
    object.type =
1
;
    object.nFleets =
nBDeciders
;
    object.teamColor =
lightSteelBlue
;
  }

  /**
   * Setups an embedded object instance<br>
   */
  private void create_influencersB_xjal(Turtle object, final int index
) {
    object.setEnvironment(
environment
);
    object.create();

    // Port connections
  }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_influencersR_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, influencersR );

    influencersR._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
```

```
   */
  private void setupParameters_influencersR_xjal(Turtle object, final
int index ) {
    object.type =
1
;
    object.nFleets =
nRDeciders
;
    object.teamColor =
red
;
  }

  /**
   * Setups an embedded object instance<br>
   */
  private void create_influencersR_xjal(Turtle object, final int index
) {
    object.setEnvironment(
environment
);
    object.create();

    // Port connections
  }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_sensorsB_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, sensorsB );

    sensorsB._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
   */
  private void setupParameters_sensorsB_xjal(Turtle object, final int
index ) {
    object.type =
2
;
    object.nFleets =
nBDeciders
;
    object.teamColor =
lightSteelBlue
;
  }

  /**
   * Setups an embedded object instance<br>
   */
```

```
  private void create_sensorsB_xjal(Turtle object, final int index ) {
    object.setEnvironment(
environment
);
    object.create();

    // Port connections
  }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   */
  private Turtle instantiate_sensorsR_xjal( final int index ) {
    Turtle object = new Turtle( getEngine(), this, sensorsR );

    sensorsR._add(object);

    return object;
  }

  /**
   * Setups parameters of an embedded object instance<br>
   */
  private void setupParameters_sensorsR_xjal(Turtle object, final int
index ) {
    object.type =
2
;
    object.nFleets =
nRDeciders
;
    object.teamColor =
red
;
  }

  /**
   * Setups an embedded object instance<br>
   */
  private void create_sensorsR_xjal(Turtle object, final int index ) {
    object.setEnvironment(
environment
);
    object.create();

    // Port connections
  }

  // Functions


void
 sense(  ) {

for (Turtle d: deciderB) {
      int ind = d.getIndex();
      for (Turtle s: d.inTurtles) {
```

```
            for (Turtle e: influencersR) {
                  if (s.distanceTo(e) <= sRange)
                        e.sensedBD[ind] = true;
            }
            for (Turtle e: sensorsR) {
                  if (s.distanceTo(e) <= sRange)
                        e.sensedBD[ind] = true;
            }
      }
}

for (Turtle d: deciderR) {
      int ind = d.getIndex();
      for (Turtle s: d.inTurtles) {
            for (Turtle e: influencersB) {
                  if (s.distanceTo(e) <= sRange)
                        e.sensedRD[ind] = true;
            }
            for (Turtle e: sensorsB) {
                  if (s.distanceTo(e) <= sRange)
                        e.sensedRD[ind] = true;
            }     }
}
  }


void
 track(  ) {

// THIS FUNCTION JUST SHOWS TRACKING LINKS

for (Turtle d: deciderB) {
      for (Turtle s: d.outTurtles) {
            for (Turtle e: influencersR) {
                  if (s.distanceTo(e) <= iRange)
                        s.outTurtles.add(e);
            }
            for (Turtle e: sensorsR) {
                  if (s.distanceTo(e) <= iRange)
                        s.outTurtles.add(e);
            }
      }
}

for (Turtle d: deciderR) {
      for (Turtle s: d.outTurtles) {
            for (Turtle e: influencersB) {
                  if (s.distanceTo(e) <= iRange)
                        s.outTurtles.add(e);
            }
            for (Turtle e: sensorsB) {
                  if (s.distanceTo(e) <= iRange)
                        s.outTurtles.add(e);
            }     }
}
  }
```

```
void
 shoot(  ) {

for (Turtle d: deciderB) {
      int ind = d.getIndex();
      for (Turtle s: d.outTurtles) {
            Turtle closestTarget = null;
            double closestDistance = Double.POSITIVE_INFINITY;
            for (Turtle e: s.outTurtles) {
                  if (!e.sensedBD[ind]) {
                        continue;
                  }
                  double dist = s.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }
            }
            if (closestTarget != null) {
                  closestTarget.dead = 1;

            }
      }
}
for (Turtle d: deciderR) {
      int ind = d.getIndex();
      for (Turtle s: d.outTurtles) {
            Turtle closestTarget = null;
            double closestDistance = Double.POSITIVE_INFINITY;
            for (Turtle e: s.outTurtles) {
                  if (!e.sensedRD[ind])
                        continue;
                  double dist = s.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }

            }
            if (closestTarget != null) {
                  closestTarget.dead = 1;
;
            }
      }
}
  }


void
 kill(  ) {

for (int i = influencersR.size()-1; i>=0; i--) {
      Turtle t = influencersR.get(i);
      if (t.dead == 1) {
            remove_influencersR(t);
      }
```

```
}
for (int i = influencersB.size()-1; i>=0; i--) {
      Turtle t = influencersB.get(i);
      if (t.dead == 1) {
            remove_influencersB(t);
      }
}
for (int i = sensorsR.size()-1; i>=0; i--) {
      Turtle t = sensorsR.get(i);
      if (t.dead == 1) {
            remove_sensorsR(t);
      }
}

for (int i = sensorsB.size()-1; i>=0; i--) {
      Turtle t = sensorsB.get(i);
      if (t.dead == 1) {
            remove_sensorsB(t);
      }
}

  }


void
 moveInfluencers(  ) {

for (Turtle d: deciderB) {
      int ind = d.getIndex();
      for (Turtle i: d.outTurtles) {
            Turtle closestTarget = null;
            double closestDistance = Double.POSITIVE_INFINITY;
            for (Turtle e: influencersR) {
                  if (!e.sensedBD[ind] || e.dead == 1)
                        continue;
                  double dist = i.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }
            }
            for (Turtle e: sensorsR) {
                  if (!e.sensedBD[ind] || e.dead == 1)
                        continue;
                  double dist = i.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }
            }
            // move
            if (closestTarget != null) {
                  i.setXY(i.getX() + (closestTarget.getX() -
i.getX())/closestDistance , i.getY() + (closestTarget.getY() -
i.getY())/closestDistance);
            }
      }
```

```
}
for (Turtle d: deciderR) {
      int ind = d.getIndex();
      for (Turtle i: d.outTurtles) {
            Turtle closestTarget = null;
            double closestDistance = Double.POSITIVE_INFINITY;
            for (Turtle e: influencersB) {
                  if (!e.sensedRD[ind] || e.dead == 1)
                        continue;
                  double dist = i.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }
            }
            for (Turtle e: sensorsB) {
                  if (!e.sensedRD[ind] || e.dead == 1)
                        continue;
                  double dist = i.distanceTo(e);
                  if (dist < closestDistance) {
                        closestTarget = e;
                        closestDistance = dist;
                  }
            }
            // move
            if (closestTarget != null) {
                  i.setXY(i.getX() + (closestTarget.getX() -
i.getX())/closestDistance , i.getY() + (closestTarget.getY() -
i.getY())/closestDistance);
            }
      }
}
  }


void
 moveSensors(  ) {

for (Turtle d: deciderB) {
      int ind = d.getIndex();
      boolean sensed = false;
      for (Turtle e:influencersR) {
            if (e.sensedBD[ind] && e.dead != 1) {
                  sensed = true;
                  break;
            }
      }
      if (sensed)
            continue;
      for (Turtle e:sensorsR) {
            if (e.sensedBD[ind] && e.dead != 1) {
                  sensed = true;
                  break;
            }
      }
      if (sensed)
            continue;
```

```
        for (Turtle s: d.inTurtles) {
              Turtle closestTarget = null;
              double closestDistance = Double.POSITIVE_INFINITY;
              for (Turtle e: influencersR) {
                    if (e.sensedBD[ind] || e.dead == 1)
                          continue;
                    double dist = s.distanceTo(e);
                    if (dist < closestDistance) {
                          closestTarget = e;
                          closestDistance = dist;
                    }
              }
              for (Turtle e: sensorsR) {
                    if (e.sensedBD[ind] || e.dead == 1)
                          continue;
                    double dist = s.distanceTo(e);
                    if (dist < closestDistance) {
                          closestTarget = e;
                          closestDistance = dist;
                    }
              }
              // move
              if (closestTarget != null) {
                    s.setXY(s.getX() + (closestTarget.getX() -
s.getX())/closestDistance , s.getY() + (closestTarget.getY() -
s.getY())/closestDistance);
              }
        }
}
for (Turtle d: deciderR) {
        int ind = d.getIndex();
        boolean sensed = false;
        for (Turtle e:influencersB) {
              if (e.sensedRD[ind] && e.dead != 1) {
                    sensed = true;
                    break;
              }
        }
        if (sensed)
              continue;
        for (Turtle e:sensorsB) {
              if (e.sensedRD[ind] && e.dead != 1) {
                    sensed = true;
                    break;
              }
        }
        if (sensed)
              continue;

        for (Turtle s: d.inTurtles) {
              Turtle closestTarget = null;
              double closestDistance = Double.POSITIVE_INFINITY;
              for (Turtle e: influencersB) {
                    if (e.sensedRD[ind] || e.dead == 1)
                          continue;
                    double dist = s.distanceTo(e);
```

```
                    if (dist < closestDistance) {
                          closestTarget = e;
                          closestDistance = dist;
                    }
              }
              for (Turtle e: sensorsB) {
                    if (e.sensedRD[ind] || e.dead == 1)
                          continue;
                    double dist = s.distanceTo(e);
                    if (dist < closestDistance) {
                          closestTarget = e;
                          closestDistance = dist;
                    }
              }
              // move
              if (closestTarget != null) {
                    s.setXY(s.getX() + (closestTarget.getX() -
s.getX())/closestDistance , s.getY() + (closestTarget.getY() -
s.getY())/closestDistance);
              }
        }
}
  }


void
 reset(  ) {

for (Turtle t: influencersR) {
      Arrays.fill(t.sensedRD, false);
      Arrays.fill(t.sensedBD, false);
      t.outTurtles.clear();
      t.inTurtles.clear();
}
for (Turtle t: influencersB) {
      Arrays.fill(t.sensedRD, false);
      Arrays.fill(t.sensedBD, false);
      t.outTurtles.clear();
      t.inTurtles.clear();
}
for (Turtle t: sensorsR) {
      Arrays.fill(t.sensedRD, false);
      Arrays.fill(t.sensedBD, false);
      t.outTurtles.clear();
      t.inTurtles.clear();
}
for (Turtle t: sensorsB) {
      Arrays.fill(t.sensedRD, false);
      Arrays.fill(t.sensedBD, false);
      t.outTurtles.clear();
      t.inTurtles.clear();
}
for (Turtle t: deciderB) {
      for (int i = t.outTurtles.size()-1;i>=0;i--) {
            Turtle k = t.outTurtles.get(i);
            if (k.getIndex()==-1)
                  t.outTurtles.remove(i);
```

```
      }
}
for (Turtle t: deciderR) {
      for (int i = t.outTurtles.size()-1;i>=0;i--) {
            Turtle k = t.outTurtles.get(i);
            if (k.getIndex()==-1)
                  t.outTurtles.remove(i);
      }
}
  }
  static final Color _rectangle_FillColor = new Color( 0xFFEFF9FE, true
);
  static final int _rectangle = 1;
  static final int influencersB_Presentation = 2;
  static final int sensorsB_Presentation = 3;
  static final int influencersR_Presentation = 4;
  static final int sensorsR_Presentation = 5;
  static final int deciderR_Presentation = 6;
  static final int deciderB_Presentation = 7;

  /**
   * Top-level presentation group id
   */
  static final int _presentation = 0;

  /**
   * Top-level icon group id
   */
  static final int _icon = -1;



  @Override
  public String getNameOfShape( int _shape ) {
    switch( _shape ) {
      case influencersB_Presentation: return
"influencersB_Presentation";
      case sensorsB_Presentation: return "sensorsB_Presentation";
      case influencersR_Presentation: return
"influencersR_Presentation";
      case sensorsR_Presentation: return "sensorsR_Presentation";
      case deciderR_Presentation: return "deciderR_Presentation";
      case deciderB_Presentation: return "deciderB_Presentation";
      default: return super.getNameOfShape( _shape );
    }
  }

  @Override
  public int getShapeType( int _shape ) {
    switch( _shape ) {
      case influencersB_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case sensorsB_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case influencersR_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case sensorsR_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case deciderR_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case deciderB_Presentation: return SHAPE_EMBEDDED_OBJECT;
      default: return super.getShapeType( _shape );
```

```
    }
  }

  @Override
  public int getShapeReplication( int _shape ) {
    switch( _shape ) {
      case influencersB_Presentation: return
influencersB.size()
;
      case sensorsB_Presentation: return
sensorsB.size()
;
      case influencersR_Presentation: return
influencersR.size()
;
      case sensorsR_Presentation: return
sensorsR.size()
;
      case deciderR_Presentation: return
deciderR.size()
;
      case deciderB_Presentation: return
deciderB.size()
;
      default: return super.getShapeReplication( _shape );
    }
  }

  @Override
  public double getShapeX( int _shape, int index ) {
    switch( _shape ) {
      case influencersB_Presentation: return 40;
      case sensorsB_Presentation: return 40;
      case influencersR_Presentation: return 40;
      case sensorsR_Presentation: return 40;
      case deciderR_Presentation: return 40;
      case deciderB_Presentation: return 40;
      default: return super.getShapeX( _shape, index );
    }
  }

  @Override
  public double getShapeY( int _shape, int index ) {
    switch( _shape ) {
      case influencersB_Presentation: return 40;
      case sensorsB_Presentation: return 40;
      case influencersR_Presentation: return 40;
      case sensorsR_Presentation: return 40;
      case deciderR_Presentation: return 40;
      case deciderB_Presentation: return 40;
      default: return super.getShapeY( _shape, index );
    }
  }

  @Override
  public Object getShapeEmbeddedObject( int _shape ) {
    switch( _shape ) {
```

```
      case deciderB_Presentation: return deciderB;
      case deciderR_Presentation: return deciderR;
      case influencersB_Presentation: return influencersB;
      case influencersR_Presentation: return influencersR;
      case sensorsB_Presentation: return sensorsB;
      case sensorsR_Presentation: return sensorsR;
      default: return super.getShapeEmbeddedObject( _shape );
    }
  }



  ShapeRectangle rectangle;

  // Static initialization of persistent elements
  {
    rectangle = new ShapeRectangle(
       true,0, 0, 0.0,
            black, _rectangle_FillColor,
                  180, 180,
                  1, LINE_STYLE_SOLID
    );
  }
  ShapeGroup presentation;
  ShapeGroup icon;

  @Override
  public Object getPersistentShape( int _shape ) {
    switch(_shape){
      case _presentation: return presentation;
      case _icon: return icon;

      case _rectangle: return rectangle;
      default: return null;
    }
  }


  @Override
  public void drawModelElements(Panel _panel, Graphics2D _g, boolean
_publicOnly ) {
    if (!_publicOnly) {
      drawEvent( _panel, _g, 780, 70, 10, 0, "event", event );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 270, 50, 10, 0, "nBDeciders",
nBDeciders, false, false );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 450, 110, 10, 0, "sRange", sRange,
false, false );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 450, 130, 10, 0, "iRange", iRange,
false, false );
```

```
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 450, 50, 10, 0, "BID", BID, false,
false );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 450, 70, 10, 0, "RID", RID, false,
false );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 450, 90, 10, 0, "seed", seed, false,
false );
    }
    if (!_publicOnly) {
      drawParameter( _panel, _g, 270, 70, 10, 0, "nRDeciders",
nRDeciders, false, false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 640, 160, 10, 0, "tick", tick,
false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 640, 180, 10, 0, "result", result,
false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 780, 20, 10, 0, "bWin", bWin,
false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 780, 40, 10, 0, "rWin", rWin,
false );
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 20, 10, 0, "sense");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 40, 10, 0, "track");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 60, 10, 0, "shoot");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 80, 10, 0, "kill");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 100, 10, 0, "moveInfluencers");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 120, 10, 0, "moveSensors");
    }
    if (!_publicOnly) {
      drawFunction( _panel, _g, 640, 140, 10, 0, "reset");
    }
    // Embedded object "deciderB"
    if (!_publicOnly) {
```

```
      drawEmbeddedObjectModelDefault( _panel, _g, 270 , 150 , -19, -21,
"deciderB", this.deciderB );
    }
    // Embedded object "deciderR"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 360 , 150 , -19, -21,
"deciderR", this.deciderR );
    }
    // Embedded object "influencersB"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 270 , 210 , -19, -21,
"influencersB", this.influencersB );
    }
    // Embedded object "influencersR"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 360 , 210 , -19, -21,
"influencersR", this.influencersR );
    }
    // Embedded object "sensorsB"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 270 , 260 , -19, -21,
"sensorsB", this.sensorsB );
    }
    // Embedded object "sensorsR"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 360 , 260 , -19, -21,
"sensorsR", this.sensorsR );
    }
    if (!_publicOnly) {
      drawEnvironment( _panel, _g, 270, 20, 10, 0, "environment",
environment );
    }
  }

  @Override
  public boolean onClickModelAt( Panel panel, double x, double y, int
clickCount, boolean publicOnly ) {
    if( !publicOnly && modelElementContains(x, y, 270, 50) ) {
      panel.addInspect( 270, 50, this, "nBDeciders" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 450, 110) ) {
      panel.addInspect( 450, 110, this, "sRange" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 450, 130) ) {
      panel.addInspect( 450, 130, this, "iRange" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 450, 50) ) {
      panel.addInspect( 450, 50, this, "BID" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 450, 70) ) {
      panel.addInspect( 450, 70, this, "RID" );
      return true;
    }
```

```
    if( !publicOnly && modelElementContains(x, y, 450, 90) ) {
      panel.addInspect( 450, 90, this, "seed" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 270, 70) ) {
      panel.addInspect( 270, 70, this, "nRDeciders" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 640, 160) ) {
      panel.addInspect( 640, 160, this, "tick" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 640, 180) ) {
      panel.addInspect( 640, 180, this, "result" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 780, 20) ) {
      panel.addInspect( 780, 20, this, "bWin" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 780, 40) ) {
      panel.addInspect( 780, 40, this, "rWin" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 780, 70) ) {
      panel.addInspect( 780, 70, this, "event" );
      return true;
    }
    if( !publicOnly && modelElementContains(x, y, 270, 20) ) {
      panel.addInspect( 270, 20, this, "environment" );
      return true;
    }
    if ( !deciderB.isEmpty() && modelElementContains(x, y, 270, 150) )
{
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 270, 150, this, "deciderB" );
      } else {
        panel.addInspect( 270, 150, this, "deciderB" );
      }
      return true;
    }
    if ( !deciderR.isEmpty() && modelElementContains(x, y, 360, 150) )
{
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 360, 150, this, "deciderR" );
      } else {
        panel.addInspect( 360, 150, this, "deciderR" );
      }
      return true;
    }
    if ( !influencersB.isEmpty() && modelElementContains(x, y, 270,
210) ) {
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 270, 210, this, "influencersB" );
      } else {
        panel.addInspect( 270, 210, this, "influencersB" );
      }
```

```
      return true;
    }
    if ( !influencersR.isEmpty() && modelElementContains(x, y, 360,
210) ) {
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 360, 210, this, "influencersR" );
      } else {
        panel.addInspect( 360, 210, this, "influencersR" );
      }
      return true;
    }
    if ( !sensorsB.isEmpty() && modelElementContains(x, y, 270, 260) )
{
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 270, 260, this, "sensorsB" );
      } else {
        panel.addInspect( 270, 260, this, "sensorsB" );
      }
      return true;
    }
    if ( !sensorsR.isEmpty() && modelElementContains(x, y, 360, 260) )
{
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 360, 260, this, "sensorsR" );
      } else {
        panel.addInspect( 360, 260, this, "sensorsR" );
      }
      return true;
    }
    return false;
  }

  // Environments
  public final Environment environment = new Environment( this );


  /**
   * Constructor
   */
  public Main( Engine engine, ActiveObject owner,
ActiveObjectCollection<? extends Main> collection ) {
    super( engine, owner, collection );
  }

  @Override
  public void create() {
    // Creating embedded object instances
    for ( int i = 0; i <
nBDeciders
; i++ ) {
      instantiate_deciderB_xjal( i );
    }
    for ( int i = 0; i <
nRDeciders
; i++ ) {
      instantiate_deciderR_xjal( i );
    }
```

```
    for ( int i = 0; i <
0
; i++ ) {
      instantiate_influencersB_xjal( i );
    }
    for ( int i = 0; i <
0
; i++ ) {
      instantiate_influencersR_xjal( i );
    }
    for ( int i = 0; i <
0
; i++ ) {
      instantiate_sensorsB_xjal( i );
    }
    for ( int i = 0; i <
0
; i++ ) {
      instantiate_sensorsR_xjal( i );
    }
    // Assigning initial values for plain variables
    bWin =
0
;
    rWin =
0
;
    // Dynamic initialization of persistent elements
    presentation = new ShapeGroup( Main.this, true, 0, 0, 0, rectangle,
influencersB_Presentation, sensorsB_Presentation,
influencersR_Presentation, sensorsR_Presentation,
deciderR_Presentation, deciderB_Presentation );
    icon = new ShapeGroup( Main.this, true, 0, 0, 0
);
    // Environments setup
    environment.disableSteps();
    environment.setSpaceContinuous(
100 ,
100  );
    environment.setNetworkUserDefined();
    environment.setLayoutType( Environment.LAYOUT_RANDOM );
       // Port connectors with non-replicated objects
    // Creating replicated embedded objects
    for ( int i = 0; i < deciderB.size(); i++ ) {
       setupParameters_deciderB_xjal( deciderB.get(i), i );
       create_deciderB_xjal( deciderB.get(i), i );
    }
    for ( int i = 0; i < deciderR.size(); i++ ) {
       setupParameters_deciderR_xjal( deciderR.get(i), i );
       create_deciderR_xjal( deciderR.get(i), i );
    }
    for ( int i = 0; i < influencersB.size(); i++ ) {
       setupParameters_influencersB_xjal( influencersB.get(i), i );
       create_influencersB_xjal( influencersB.get(i), i );
    }
    for ( int i = 0; i < influencersR.size(); i++ ) {
       setupParameters_influencersR_xjal( influencersR.get(i), i );
```

```
        create_influencersR_xjal( influencersR.get(i), i );
      }
      for ( int i = 0; i < sensorsB.size(); i++ ) {
        setupParameters_sensorsB_xjal( sensorsB.get(i), i );
        create_sensorsB_xjal( sensorsB.get(i), i );
      }
      for ( int i = 0; i < sensorsR.size(); i++ ) {
        setupParameters_sensorsR_xjal( sensorsR.get(i), i );
        create_sensorsR_xjal( sensorsR.get(i), i );
      }
      assignInitialConditions();
      onCreate();
    }

    @Override
    public void start() {
      event.start();
      environment.applyLayout();
      for (ActiveObject embeddedObject : deciderB){
        embeddedObject.start();
      }
      for (ActiveObject embeddedObject : deciderR){
        embeddedObject.start();
      }
      for (ActiveObject embeddedObject : influencersB){
        embeddedObject.start();
      }
      for (ActiveObject embeddedObject : influencersR){
        embeddedObject.start();
      }
      for (ActiveObject embeddedObject : sensorsB){
        embeddedObject.start();
      }
      for (ActiveObject embeddedObject : sensorsR){
        embeddedObject.start();
      }
      onStartup();
    }

    public void onStartup() {
      super.onStartup();

  for (int i = 0; i < nBDeciders; i++) {
        for (int j = 0; j < cB[BID][i]; j++) {
              Turtle t = add_sensorsB();
              t.sensedRD = new boolean[nRDeciders];
              t.decider = deciderB.get(i);
              deciderB.get(i).inTurtles.add(t);
        }
  }
  for (int i = 0; i < nBDeciders; i++) {
        for (int j = 0; j < cB[BID][nBDeciders + i]; j++) {
              Turtle t = add_influencersB();
              t.sensedRD = new boolean[nRDeciders];
              t.decider = deciderB.get(i);
              deciderB.get(i).outTurtles.add(t);
        }
```

```
}
for (int i = 0; i < nRDeciders; i++) {
      for (int j = 0; j < cR[RID][i]; j++) {
            Turtle t = add_sensorsR();
            t.sensedBD = new boolean[nBDeciders];
            t.decider = deciderB.get(i);
            deciderR.get(i).inTurtles.add(t);
      }
}
for (int i = 0; i < nRDeciders; i++) {
      for (int j = 0; j < cR[RID][nRDeciders + i]; j++) {
            Turtle t = add_influencersR();
            t.sensedBD = new boolean[nBDeciders];
            t.decider = deciderB.get(i);
            deciderR.get(i).outTurtles.add(t);
      }
}


  }

  public List<Object> getEmbeddedObjects() {
    LinkedList<Object> list = new LinkedList<Object>();
    list.add( deciderB );
    list.add( deciderR );
    list.add( influencersB );
    list.add( influencersR );
    list.add( sensorsB );
    list.add( sensorsR );
    return list;
  }

  public void onDestroy() {
    super.onDestroy();
    event.onDestroy();
    environment.onDestroy();
    for (ActiveObject embeddedObject : deciderB) {
      embeddedObject.onDestroy();
    }
    for (ActiveObject embeddedObject : deciderR) {
      embeddedObject.onDestroy();
    }
    for (ActiveObject embeddedObject : influencersB) {
      embeddedObject.onDestroy();
    }
    for (ActiveObject embeddedObject : influencersR) {
      embeddedObject.onDestroy();
    }
    for (ActiveObject embeddedObject : sensorsB) {
      embeddedObject.onDestroy();
    }
    for (ActiveObject embeddedObject : sensorsR) {
      embeddedObject.onDestroy();
    }
  }

  // Additional class code
```

```
int[][] cB = {
{4,3,3,2,1,4,1,6},
{4,3,3,2,1,3,3,5},
{4,3,3,2,1,3,2,6},
{4,3,3,2,1,3,1,7},
.
.
.
.

{3,3,3,3,5,4,2,1},
{3,3,3,3,5,3,3,1},
{3,3,3,3,5,3,2,2},
{3,3,3,3,4,4,3,1},
{3,3,3,3,4,4,2,2},
{3,3,3,3,4,3,3,2},
{3,3,3,3,3,3,3,3}
};
int[][] cR = {
{9,1,1,1,9,1,1,1},
{9,1,1,1,8,2,1,1},
{9,1,1,1,7,3,1,1},
{9,1,1,1,7,2,2,1},
{9,1,1,1,6,4,1,1},
{9,1,1,1,6,3,2,1},
{9,1,1,1,6,2,2,2},
{9,1,1,1,5,5,1,1},
.
.
.
.
.
{9,1,1,1,2,4,3,3},
{9,1,1,1,1,9,1,1},
{9,1,1,1,1,8,2,1},
{9,1,1,1,1,7,3,1},
{9,1,1,1,1,7,2,2},
{9,1,1,1,1,6,4,1},
{9,1,1,1,1,6,3,2},
{3,3,3,3,4,4,3,1},
{3,3,3,3,4,4,2,2},
{3,3,3,3,4,3,3,2},
{3,3,3,3,3,3,3,3}
};
  // End of additional class code

}
```

APPENDIX C: MATLAB CODES TO CALCULATE THE EIGENVALUES OF DIFFERENT MEANINGFUL COMBINATIONS

In this appendix, two Matlab codes for 4-Decider and 6-Decider are just presented to calculate the eigenvalues of Decider basis different meaningful combinations. These Matlab codes give the rationale of how it works for different numbers of Deciders.

C.1. MATLAB CODE TO CALCULATE EIGENVALUES OF DIFFERENT MEANINGFUL COMBINATIONS OF 4 DECIDERS.

```
clc
clear all
format ('short');
loadfile = 'C:\Documents and Settings\mfida001\My Documents\MATLAB\InPuts\InPutX_4\6_4_6.txt';
a=load(loadfile);
b=size(a);
c=0;
for j=1:b(1,2)/2;
c=c+a(1,j);
end;
for i=1:b(1,1);
v1=[ones(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1)];
v2=[zeros(a(i,1),1);ones(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1)];
v3=[zeros(a(i,1),1);zeros(a(i,2),1);ones(a(i,3),1);zeros(a(i,4),1)];
v4=[zeros(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);ones(a(i,4),1)];
g1=[v1,v2,v3,v4];
v5=[ones(a(i,5),1);zeros(a(i,6),1);zeros(a(i,7),1);zeros(a(i,8),1)];
v6=[zeros(a(i,5),1);ones(a(i,6),1);zeros(a(i,7),1);zeros(a(i,8),1)];
v7=[zeros(a(i,5),1);zeros(a(i,6),1);ones(a(i,7),1);zeros(a(i,8),1)];
v8=[zeros(a(i,5),1);zeros(a(i,6),1);zeros(a(i,7),1);ones(a(i,8),1)];
g2=[v5,v6,v7,v8];
h=zeros(c);
m=zeros(c,1);
n=zeros(b(1,2)/2,c);
```

```
o=zeros(b(1,2)/2);
p=zeros(b(1,2)/2,1);
q=ones(c,1);
r=0;
s=[h,g1,h,m;n,o,g2',p;h,n',h,q;q',p',m',r];
t(:,i)=(eig(s));
t_real(:,i) = real(t(:,i));
t_imag(:,i) = imag(t(:,i));
Pfe(i)=max(abs(eig(s)));
Var(i)=var(t(:,i));
end;

xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_4\6_4_6.xls',
t_real','real_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_4\6_4_6.xls',
t_imag','imag_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_4\6_4_6.xls',
Pfe','PFE_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_4\6_4_6.xls',
Var','Variance');
```

C.2. MATLAB CODE TO CALCULATE EIGENVALUES OF DIFFERENT MEANINGFUL COMBINATIONS OF 4 DECIDERS.

```
clc
clear all
format ('short');
loadfile = 'C:\Documents and Settings\mfida001\My Documents\MATLAB\InPuts\InPutX_6\6_6_6.txt';
a=load(loadfile);
b=size(a);
c=0;
for j=1:b(1,2)/2;
c=c+a(1,j);
end;
for i=1:b(1,1);
v1=[ones(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1);zeros(a(i,5),1);zeros(a(i,6),1)];
v2=[zeros(a(i,1),1);ones(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1);zeros(a(i,5),1);zeros(a(i,6),1)];
v3=[zeros(a(i,1),1);zeros(a(i,2),1);ones(a(i,3),1);zeros(a(i,4),1);zeros(a(i,5),1);zeros(a(i,6),1)];
v4=[zeros(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);ones(a(i,4),1);zeros(a(i,5),1);zeros(a(i,6),1)];
v5=[zeros(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1);ones(a(i,5),1);zeros(a(i,6),1)];
v6=[zeros(a(i,1),1);zeros(a(i,2),1);zeros(a(i,3),1);zeros(a(i,4),1);zeros(a(i,5),1);ones(a(i,6),1)];
g1=[v1,v2,v3,v4,v5,v6];
v7=[ones(a(i,7),1);zeros(a(i,8),1);zeros(a(i,9),1);zeros(a(i,10),1);zeros(a(i,11),1);zeros(a(i,12),1)
];
v8=[zeros(a(i,7),1);ones(a(i,8),1);zeros(a(i,9),1);zeros(a(i,10),1);zeros(a(i,11),1);zeros(a(i,12),1)
];
v9=[zeros(a(i,7),1);zeros(a(i,8),1);ones(a(i,9),1);zeros(a(i,10),1);zeros(a(i,11),1);zeros(a(i,12),1)
];
v10=[zeros(a(i,7),1);zeros(a(i,8),1);zeros(a(i,9),1);ones(a(i,10),1);zeros(a(i,11),1);zeros(a(i,12),1
)];
v11=[zeros(a(i,7),1);zeros(a(i,8),1);zeros(a(i,9),1);zeros(a(i,10),1);ones(a(i,11),1);zeros(a(i,12),1
)];
v12=[zeros(a(i,7),1);zeros(a(i,8),1);zeros(a(i,9),1);zeros(a(i,10),1);zeros(a(i,11),1);ones(a(i,12),1
)];
g2=[v7,v8,v9,v10,v11,v12];
h=zeros(c);
m=zeros(c,1);
n=zeros(b(1,2)/2,c);
o=zeros(b(1,2)/2);
p=zeros(b(1,2)/2,1);
q=ones(c,1);
```

```
r=0;
s=[h,g1,h,m;n,o,g2',p;h,n',h,q;q',p',m',r];
t(:,i)=(eig(s));
t_real(:,i) = real(t(:,i));
t_imag(:,i) = imag(t(:,i));
Pfe(i)=max(abs(eig(s)));
Var(i)=var(t(:,i));
end;

xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_6\6_6_6.xls',
t_real','real_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_6\6_6_6.xls',
t_imag','imag_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_6\6_6_6.xls',
Pfe','PFE_eigenvalues');
xlswrite('C:\Documents and Settings\mfida001\My Documents\MATLAB\OutPuts\OutPutX_6\6_6_6.xls',
Var','Variance');
```

VITA

Major Mehmet Fidanci was born in Kayseri, Turkey on March 20, 1971. He graduated from the Turkish Air Force Academy, in Istanbul, Turkey in 1993 with a B.S. in Aeronautical Engineering and was commissioned as a Second Lieutenant to the Undergraduate Pilot Training Wing Command. He received a M.S. in Systems Engineering from the Air Force Institute of Technology (AFIT) Dayton, Ohio in 2000. He performed this research while a student of the Department of Engineering Management and Systems Engineering, Old Dominion University, in Norfolk, Virginia.

During his 17 years of active service, he trained for undergraduate pilot training for two years, then he served as a radar interceptor for almost two and a half years. Then, he taught Probabilistic Theory and Statistics in the Department of Industrial Engineering in Turkish Air Force Academy for seven years. He has also served as a research and development officer in the Department of Planning, Evaluation and Research & Development and Scheduler Officier in Aeronautics and Space Technologies Institute during his years in the Turkish Air Force Academy.