A SYSTEM-OF-SYSTEMS FLEXIBILITY FRAMEWORK: A METHOD FOR EVALUATING DESIGNS THAT ARE SUBJECTED TO DISRUPTIONS

A Thesis
Presented to
The Academic Faculty

by

David S Warshawsky

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

Georgia Institute of Technology
December 2015

A SYSTEM-OF-SYSTEMS FLEXIBILITY FRAMEWORK: A METHOD FOR EVALUATING DESIGNS THAT ARE SUBJECTED TO DISRUPTIONS

Approved by:

Professor Dimitri N Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Professor Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Daniel Cooksey
School of Aerospace Engineering
Georgia Institute of Technology

Professor John Salmon
School of Aerospace Engineering
Georgia Institute of Technology

Professor Graeme Kennedy
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: August 20, 2015

To Adina, for always believing in me

ACKNOWLEDGEMENTS

First I want to thank my advisor Dr. Dimitri Mavris for giving me the chance to excel in a PHD program. I originally had my heart set on studying control systems but he convinced me to leave my comfort zone and explore the wide world of systems engineering. I am extremely grateful for his tireless effort in running the Aerospace Systems Design Lab and obtaining enough funding so that all of us students could work on real world research projects and gain valuable engineering experience.

Next I want to thank Kelly Griendling for singling me out from the throng of first years and introducing me to the new and growing field of systems of systems. Through her direction, I found a topic that was both interesting to me and worth writing a dissertation on. I am especially grateful for the friendly and supportive environment that she created in the ARCHITECT work group that gave me confidence and a feeling of belonging. I also want to thank Daniel Cooksey for always being there to listen to my crazy ideas and to keep me on track to graduate. If it weren't for his direction, I would still be sitting at my desk with my head in the clouds dreaming of cool new methods to try out, and this dissertation would never have been written.

I also want to thank the rest of my thesis committee, Dr. Kennedy, Dr. Salmon, and Dr. Schrage for taking interest in my work and giving me valuable feedback along the way.

Nothing was more valuable to me in surviving this process than the close friends I made along the way. I want to thank Aharon for befriending me as soon as I stepped foot on campus, and Jamie for all the Friday morning games of bowling where I vented my frustration. I want to thank Yaakov for always giving a sympathetic ear, and Eli for encouraging me to take breaks every once and a while. Lastly, I want to thank

the Broyde family for treating me like a son and always making sure I had a home to go to when I needed it most.

Most importantly, I cannot express my gratitude enough to my wife, Adina. She stuck with me through all the tough times and always saw the best in me, even when I was stressed or too tired to care. Thank you for tolerating the late nights and lame dinners. I owe you more than I can ever repay.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nomenclature

General Acronyms

SoS	System of systems
O&S	Operations and support
HOST	Hardware Open Systems Technologies
COTS	Commercial Off The Shelf
FACE	Future Airborne Capability Environment
LoRA	Level of Repair Analysis
ELoRA	Economic Level of Repair Analysis
FLoRA	Fleet-wide Level of Repair Analysis
M&S	Modeling and Simulations
DES	Discrete Event Simulation
EA	Evolutionary Algorithm
GA	Genetic Algorithm
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
COMPASS	Computerized Optimization Model for Predicting and Analyzing Support Structures
SRU	shop replacement unit
LRU	line replacement unit
ISR	Intelligence Surveillance and Reconnaissance
SEAD	Suppression of Enemy Air Defenses
C2	Command and Control
ARCHITECT	Architecture Based Technology Evaluation and Capability Tradeoff
RAAM	Rapid Architecture Alternative Modeling

ARCNET	Architecture Resource-Based Collaborative Network Evaluation Tool
TSP	Traveling Salesman Problem
DODAF	Department of Defense Architecture Framework
DOE	Design of Experiments
JPDF	Joint Probability Distribution Function
CDF	Cumulative Distribution Function
PCA	Principal Components Analysis

LoRA models

C_i	the set of components at indenture level i
Γ_c	the set of subcomponents of item c
g	a set of components that share a fixed cost
E	the set of echelon levels
D_e	the set of possible decisions available at echelon level e
$vc_{c,e,d}$	the variable cost associated with component c at echelon e to perform action d
$fc_{g,e,d}$	the fixed cost associated with component set G at echelon e to perform action d
γ_c	the total annual failures of component c
$X_{c,e,d}$	1 if action d is performed for component c at echelon e
$Y_{g,e,d}$	: 1 if for any component in G, action d is performed at echelon e
I	the set of indenture levels
J	the set of components
R	the set of available repair options
Γ	a set hierarchical relationships between component set X

$m(r, i)$	1 if repair option r is selected at indenture level i
$n(r, x)$	1 if repair option r is selected for component x
$c_v(r, x)$	variable cost for using repair option r for component x
$c_f(r, i)$	fixed cost for enabling repair option r at indenture level i

Maintenance Logistics Model

AMC&D	Advanced Mission Computer and Displays
JAM	Joint Aviation Model
DH	Display Head
MP	Mission processor
P	Platform
C	Component
D	Depot
I	Intermediate
W	Warehouse
O	Operator
FH/YR	Flight hours per year
CND	Cannot Duplicate
T	Time
A	Overall system availability
a_n	long term availability of platform n
a_k	availability of component n at time t
A_k	overall availability after event k
Δt_k	duration of event k

Network model

AC	Algebraic Connectivity

FV	Fiedler Vector
FC	Functional Cyclicity
PFE	Perron Frobenius Eigenvector
CNE	Coefficient of Network Effects
GE	Graph Energy
CPL	Characteristic Path Length
PCANS	Precedence Commitment Assignment Network and Skills
L_i	The load on edge i
P	A set of paths that accomplish the same capability
p_i	The number of paths that use edge i
p_i^o	the number of paths in P that don't use edge i
F_v	the probability that the demand exceeds the maximum capacity
T	the set of tasks
G	the set of agents
P	the network relating tasks to one another
P_{ij}	an edge weight in the P matrix
A	the network relating tasks to agents
A_{tg}	an edge weight in the A matrix
A_{ti}^d	the degree of a task in the A matrix
A_{gi}^d	the degree of an agent in the A matrix
N	the network relating agents to one another
N_{ij}	an edge weight in the N matrix
MF	maximum flow through the P network
Deg	maximum degree in the N1 network
Npath	number of paths in the N2 network
Cpath	most critical path in the N2 network

Pconn2	path connectivity of the N2 network
Pconn3	path connectivity of the N3 network

Optimization

NSGAII	Non-dominated Sorting Genetic Algorithm II
NSGM	NSGAII Program in Matlab
C	Cost
A	Availability
F	Flexibility
F1	Divisibility
F2	Growth flexibility
F3	Volume flexibility

SUMMARY

As systems become more interconnected due to advancements in information technology and embedded computing, the set of existing systems that a new system must interact with becomes increasingly large. These systems of systems (SoS) add additional layers of complexity to the traditional systems engineering design methodologies by exhibiting emergent, evolutionary and adaptive behavior. At the same time the life span of the SoS far exceeds the life span of its component systems and likely the operational environment as well. As such it is very difficult to accurately predict how the SoS will perform and what it will look like in the distant future. However, it is imperative that the engineers not ignore this fact when planning for the maintenance of systems within systems of systems. As such the primary goal of this dissertation will be to develop a method for SoS maintenance planning.

Without the ability to predict the changes that the SoS will be subjected to, it is not possible to accurately simulate its performance in a changing environment. Instead, it is hypothesized that by examining the system's inherent resistance to uncertainty, one can increase the performance in the presence of changes and disruptions. The literature suggests that flexibility, defined as the ability to adapt to changes, is a good attribute to measure. Therefore, the first contribution is the development of a framework for measuring the flexibility of an SoS and applying it to an example multi-platform maintenance planning problem.

Three categories of flexibility were defined that apply to systems of systems. First, volume flexibility refers to the amount of excess resources that the system possesses. It can be assumed that excess resources will be used to mitigate the negative effects

of a change in the environment. Second, divisibility is a measure of the modularity of choices in the system. The more options for performing the intended capability that are available the less likely that a disruptive event will completely disable the system. Finally, growth flexibility is a measure of how easy it is to implement a change to the system. When the system is no longer able to perform the capability due to any number of potential reasons, growth flexibility describes how much it will cost and how long it will take to get the system running again.

In order to test the hypothesis that flexible systems of systems perform better in a changing environment, an example problem was explored. Level of Repair Analysis (LoRA) is the process of planning for the replacement and repair of components in a system. Generally, this is a simple logistics optimization problem and would not be considered an SoS. However, there is a push to increase the commonality of components across multiple systems. The resulting interactions between overlapping supply chains increases the complexity of the problem and begins to resemble a simple SoS. When disruptive events are applied to the model, the desired emergent and evolutionary behaviors can be observed.

A discrete event simulation was developed and used to generate a set of randomly evaluated designs for a multi-platform maintenance scenario. For each of these designs the performance and cost was measured for a disruptive environment and an ideal one. Additionally, three measures of flexibility were evaluated corresponding to the three categories described above. It was shown that the three measures of flexibility correlate stronger with the performance and cost of the system in a disruptive environment than in an undisrupted scenario.

A second issue addressed was that current SoS simulation methods tend to be very computationally expensive. For a full scale SoS the resulting design space exploration effort could very well take several human life times to complete. Even for very simple problems the cost is significant, therefore an attempt was made to develop heuristics

that could be used to rapidly down select the design space before the expensive simulation is employed. To do this, it was assumed that all systems of systems can be described by the interaction between multiple entities or tasks. As such, it was hypothesized that a network model could always be used to describe an SoS abstractly. The advantages of network models is that relevant properties can be rapidly evaluated using simple graph theory procedures.

To test this hypothesis a network model was developed describing the interrelationships between agents and tasks in the modified LoRA problem. Fourteen network properties were identified using graph theory as having the potential to relate to the flexibility of the system. It was found that when the design variables primarily drive changes in the network topology the graph properties correlate fairly well with the flexibility measures. However when this is not the case a method was proposed for identifying the probability that a given solution will exhibit favorable performance. If that probability is high enough then it is worth spending computational resources to get an accurate assessment, otherwise it should be discarded. This method was shown to increase the average flexibility and decrease the average cost of the resulting set of down selected solutions.

Finally, it was proposed that an evolutionary algorithm would be a good method for exploring the design space efficiently. A publicly available multi-criteria genetic algorithm, NSGAII, was used to obtain the frontier of non-dominated solutions. It was found that when optimizing for flexibility the down selected subset of solutions is very close to the set of optimal solutions in the entire design space. Additionally, when heuristics were used comparable solutions were found in a fraction of the time. It should be noted that since this was a multi-attribute design problem, it is possible than any of the down selected solutions could be considered the "best" design. However, it can be definitively stated that the heuristics greatly improved the computational efficiency of the method.

CHAPTER I

BACKGROUND AND MOTIVATION

1.1 Hardware Open Systems Technologies

The Navy Hardware Open Technology Standard (HOST) was developed to increase the portability of components within Navy computer systems. The problem it is trying to solve is that the hardware standards that systems engineers must conform to when designing a new platform are flexible enough that multiple vendors can develop products that comply with the standards, meet the requirements, but are not compatible with one another. This means that when the platform design is finalized, one component must be chosen, and should something happen to that manufacturer the Navy is out of luck. Therefore, HOST is intended to reduce the variability in existing standards, so multiple different vendors can be available for a given component type.[6] Similarly, the NavAir Future Airborne Capability Environment (FACE) initiative is focused on developing modular and scalable software that maximizes reuse and interoperability across many platforms employed by the Navy. [106] In combination their future vision includes the design of a mission computer that uses more generic COTS (commercial off the shelf) components to reduce production cost, increase the ease of future upgrades, and increase the applicability to all platforms in the fleet. The desired result is an architecture for airborne computing that reduces life cycle costs and development cycle time.

Currently, when designing a new aircraft, the manufacturer is also responsible for designing the mission computer and providing replacements for the lifetime of the platform. The result is that each platform has a unique mission computer, costing hundreds of thousands of dollars with unique circuit cards costing tens of thousands

of dollars each to replace. Additionally the "core law" [43] requires that components that are mission critical must be maintained by a service organic repair depot. Combined with the cost of the parts, this means that repairs are generally done at the subcomponent level on the circuit cards themselves (integrated circuits, capacitors, etc...). This type of maintenance strategy tends to be costly, and highly skill intensive, and requires a large supply of unique spare parts.

From the perspective of operational and support (O&S) costs, the goal is to make the maintenance strategy more forgiving of the disposal of circuit cards since by using more COTS cards, purchasing new ones is cheaper and easier. Additionally because of the increase in commonality, the total number of unique spare parts that must be stored is also reduced, as well as the skill required to repair the mission computer as a whole. This could mean increasing the amount of repairs that can be done by the platform operator before it must be sent to the depot. Alternatively, HOST suggests that maintenance be performed by the mission computer manufacturers themselves, instead of a Navy operated repair depot, thereby further reducing O&S costs. Another advantage of commonality is that as upgrades become necessary, a different manufacturer can provide a better product. Then, the old boxes need not be discarded entirely due to the high level of commonality across the fleet, and the supply of spare parts need not be drastically overturned, because some other platform that did not need an upgrade yet can still use them. However, in order to realize these benefits, some amount of planning must be done to design a maintenance policy for the new mission computer. The reason for considering multiple platforms is that it allows for high level interactions between entities, such as resource sharing. This is expected to demonstrate some amount of cost savings by allowing for commonalities across platforms. For example, if two platforms use the same display head, then the depot will not need to carry as many spares. In conclusion, the paradigm shift calling for increased commonality across different platforms will complicate the process of

maintenance planning.

1.2 Internet of Things

Some of the challenges that arise from increasing the modularity of military systems can be easily illustrated with a more generic scenario relating to increased interactions between every day common items. A common theme in science fiction is a future describing data at one's fingertips and computers managing most of the mundane tasks of our everyday lives. That reality is much closer than ever before. Because of improvements in cloud data storage, wireless technology and embedded computing, many common household items are becoming more and more interconnected. The ability to know everything about anything in real time unlocks great potential in business, engineering and even day to day life.

The term "Internet of things" was coined around 20 years ago by Kevin Ashton to describe how RFID technology combined with the internet could be used in supply chain management to better track products.[12] Now "Internet of things" is used to describe everything related to overcoming the gap between physical objects and their digital representations. Anything from cars warning each other of traffic, to refrigerators tracking the expiration date on perishable items would be considered part of the Internet of things. [162]

The consequences of the desire to connect everything, is that every product now requires an embedded processor of some sort. This is not necessarily a huge problem due to the falling cost of embedded computing, and constant improvements in software. However, the issue is that in order for such a product to be successful it must have an easy to use interface, which will necessarily grow more complicated as things become more connected. Then the problem becomes, that the commonly used interfaces that people are used to have saturated the market but are closely guarded by large companies such as Microsoft, Apple, and Google. In the meantime though those

interfaces must still integrate with systems developed by other companies, which puts increased burden on those smaller developers. [57]

For example, it has been many years since car companies started introducing digital dashboard displays into automobiles. These dashboards are capable of navigation, entertainment, and connection to mobile phones to make hands-free calls. However, as mobile devices begin to perform these same functions, people begin to use their smart phone instead, since that is the interface they are more comfortable with. In response Apple is set to release a product called Carplay which is an automotive dashboard that interfaces seamlessly with the phone in a hands-free way.[10] Additionally the Open Automotive Alliance is a group of information technology and automotive companies (including Google) devoted to developing similar products, including a dashboard interface for Android devices.[5] The result is that it no longer makes sense for the car manufacturer to develop its own connected dashboard, rather it just needs to give the requirements to the mobile device manufacturers and they will develop the computer.

There is an issue here with the difference in time scales of the production and upgrades cycles between the computers and the appliances themselves. It is quite common to replace one's mobile devices every 2-3 years as newer and more capable products are released. However, the same cannot be said about cars and other appliances where the time scale can be on the order of ten to twenty years. It is then logical to consider, that in the future, people will get the connected elements of their appliances serviced and upgraded more frequently by a separate service contracted by the computer manufacturers. The ramifications of this potential paradigm shift also effect consumers that are integrally more involved in the design process. Consider how this effects a consumer that not only is paying for the product, but is also funding the research and development, such as the U.S. Navy, and is required to propose a maintenance policy for the new system up front. A maintenance policy

that must account for the fact that the new system is modular in nature and will be serviced by multiple contractors. This is the primary problem with the HOST design paradigm that this thesis will attempt to address. How does the system designer plan for the maintenance of a system that has many independently designed parts whose maintenance requirements and upgrade cycles are potentially in conflict?

1.3 Level of Repair Analysis

The first step in answering the primary research question of this thesis is to benchmark how maintenance planning is currently done. Whenever the DoD commissions a new system, it must also design a maintenance strategy to go along with it. The maintenance strategy is responsible for "delivering economical and reliable mission ready systems, sub-systems, and components to effectively and efficiently support the war-fighter"[113]. The way it is done now is a process called Level of Repair Analysis (LoRA). LoRA is the methodology for planning how and where each component of a system will be maintained at minimum cost. It is primarily based around a heuristic selection process, relying on subject matter experts (SMEs) to answer questions to evaluate alternatives. Once a set of feasible alternatives is obtained an optimizer is frequently applied to the economic criteria to finalize the maintenance architecture. This two step process can be broken down as follows.

1. Use noneconomic decision criteria to make the initial support decisions.

2. Use an economic model to optimize for the most cost effective alternative

As with the Internet of things, LoRA becomes interesting when obsolescence is introduced to the problem. The more CoTS components are used, the harder it is to choose components that will be available throughout the life span of the system, and the need to plan for upgrades becomes more important. When the system attempts to replace an obsolete component, it attempts to match the fit, form and function of

the old component. However, the more these attributes vary the harder it is likely to be to retain the maintenance strategy used by the original platform. This change in requirements will require the system to evolve. Advanced knowledge of these changes could potentially impact the original design.

1.3.1 Define terminology

[139]

- Echelon

  The echelons are the maintenance levels. Generally speaking level 1 refers to the operator and level 3 is a depot. Any number of echelons can be used, however realistically there are usually not more than four.

- Operator

  The echelon where the end items are actually used to perform tasks. As such this is the only location where failures occur. Operators are usually capable of performing limited maintenance functions though it usually costs a lot to enable this capability.

- Depot

  Depot maintenance is defined in U.S. code title 10 section 2460 [1] as "material maintenance or repair requiring the overhaul, upgrading, or rebuilding of parts, assemblies, or sub-assemblies, and the testing and reclamation of equipment as necessary, regardless of the source of funds for the maintenance or repair or the location at which the maintenance or repair is performed". For example, the Norfolk naval shipyard is a US Navy maintenance depot. Alternatively, some components also get maintained at the original equipment manufacturer. In this case that would be considered the depot.

- Intermediate level of repair

  AFI 21-101 defines intermediate level maintenance as the "second level of maintenance performed off-equipment (on removed component parts or equipment) at backshop level. Primarily testing and repair or replacement of component parts. This level also includes Centralized Intermediate Repair Facilities (CIRFs)." [154] Intermediate levels tend to serve multiple operators in close proximity to one another, and are more capable of performing maintenance than the operators. They are frequently used to perform testing on dysfunctional components that have already been removed from the platform at the operator. At which point repairs can sometimes be done at the I-level, but other times it must be sent along to a full depot.

- Indenture

  The levels of indenture refer to the hierarchy of components in the system. Level one is the platform also known as the end item. Level two is also sometimes known as a line replacement unit (LRU) and level three is sometimes referred to as a shop replacement unit (SRU).

Any number of echelons and indenture levels can be considered, but the complexity of the problem does increase with the levels considered. For this study it is assumed that three echelons and two levels of indenture are sufficient for observing meaningful behaviors performing the desired tradeoffs.

1.3.2 LoRA in practice

From the MIL STD 1390D [3] the level of repair analysis (LORA) is one of the prescribed techniques in the military and maritime industries to achieve a system design with the minimum whole life maintenance cost. It applies to all system acquisition, modification and R&D programs and is an integral part of logistics support analysis.

It influences the support cost which in turn influences the total life cycle cost of ownership. Additionally it has an important impact on the operational readiness of the system. It uses economic and non-economic criteria as well as readiness objectives to determine the least costly method for replacing, discarding or repairing each component in the system. Therefore, the goal is to minimize the maintenance cost and potentially influence the system design from the very beginning of the design process.

Early on, when the system architecture is still flexible, LoRA is used to determine the number of maintenance sites, number of systems, and the component characteristics such as the minimum MTBF (Mean Time Between Failures) that the maintenance system can handle. Later on in the system life cycle it can be used to determine whether or not it is worth repairing the components when they break. Additionally, it can be used to determine the support equipment required at the maintenance sites, the number of spare components that should be purchased, and how to schedule maintenance. Finally, later in the system's life cycle, LoRA can be used to evaluate changes that must be made to the system to determine if the upgraded system can still be maintained affordably. This perspective on LoRA gives a good idea of the types of tradeoffs that can be made with the method but is not a good documentation of how it is actually performed.

For a perspective on how LoRA is implemented, the MTAIN manual on LoRA states [87] that "The Level Of Repair Analysis (LORA) is instrumental in providing an optimized maintenance philosophy based upon a cost rational." LoRA is an iterative process part of logistics support analysis. The non-economic part is a screening process before economic LoRA is done. It is based on a set of prescribed questions and a logic chart. An example of the prescribed questions and logic chart can be found in figures 3 and 4. The problem with this step is that it requires the subjective input of subject matter experts and is not scientifically reproducible making it a poor choice of study for a thesis. Therefore, the main focus will be on the second step,

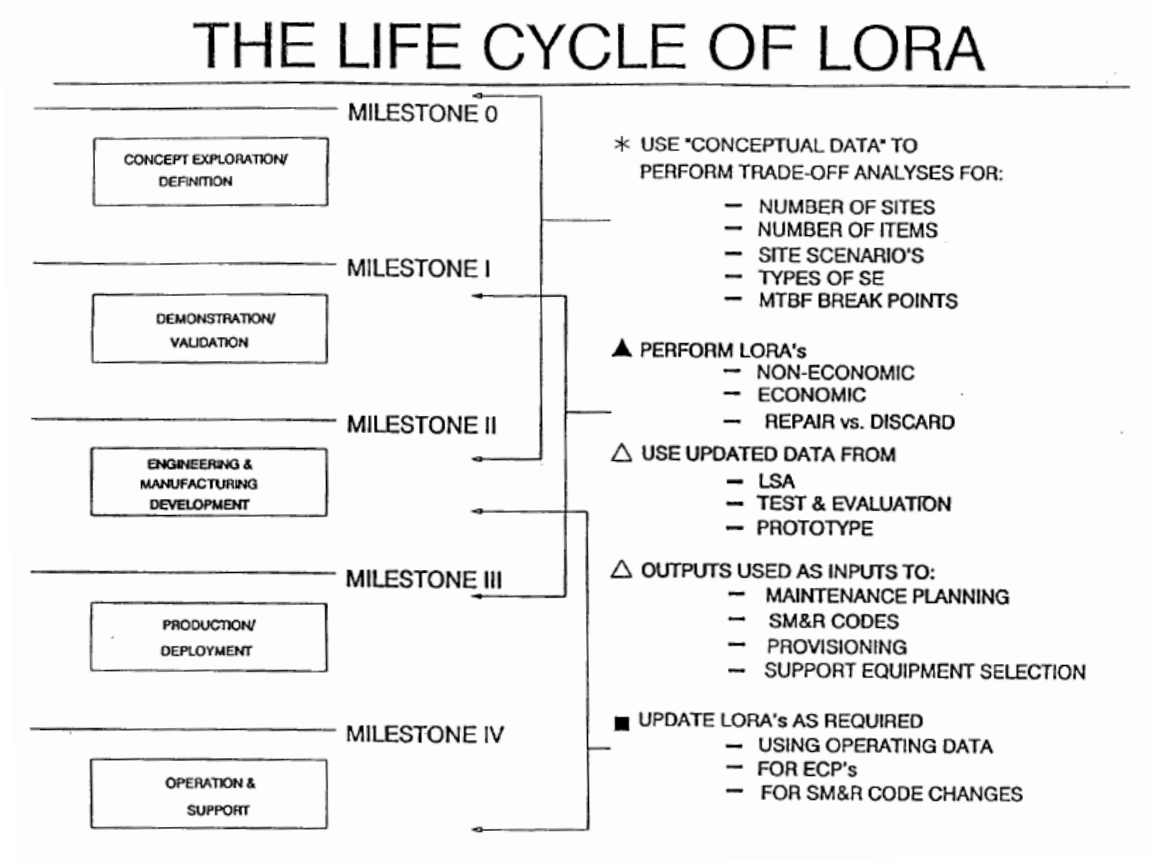


Figure 1: Purpose of LoRA during different parts of the life cycle of the system [3]

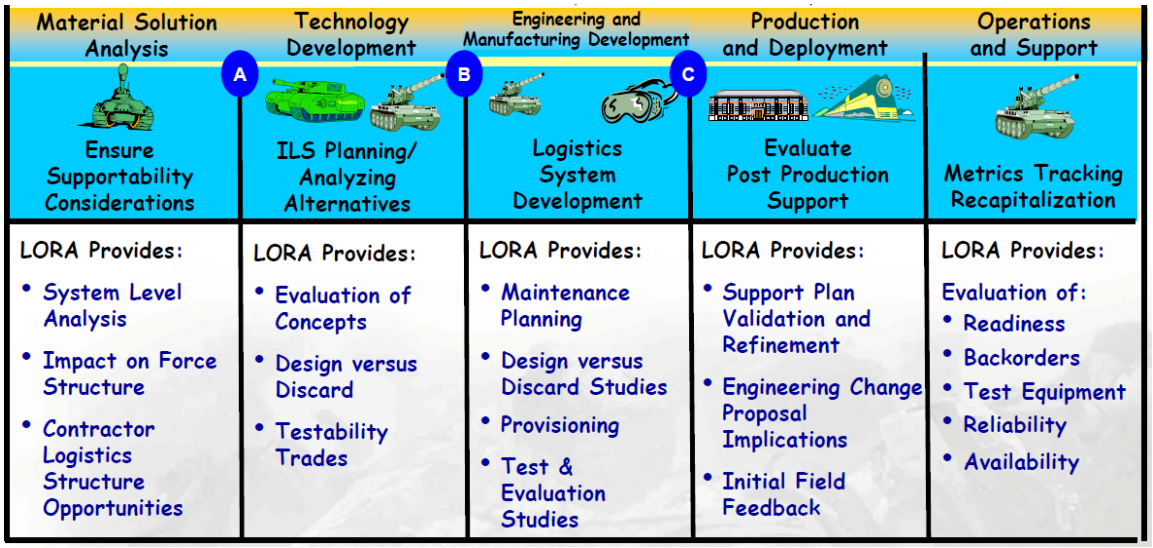


Figure 2: Purpose of LoRA during different parts of the life cycle of the system [122]

computational LoRA, which is a repeatable method.

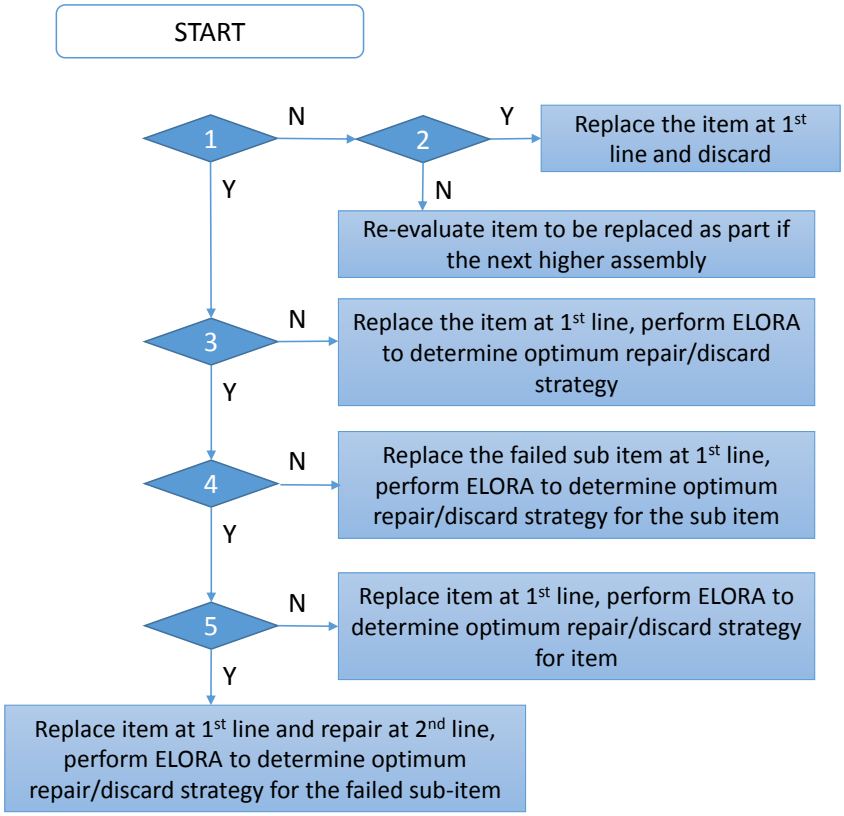


Figure 3: Example of non-economic LoRA logic flow diagram [87]

Economic LoRA is the second step and is used to fine tune the strategies that result from step 1 using computational models and optimization algorithms. To quote the manual, "these algorithms require specific input data for the system inherent supportability characteristics and its intended "Use" or deployment profile. System level data include the system's expected operational life, operational profile, and site data (distances between operational centers and repair shop, including the total number of these facilities.). The item data include component MTBF, cost, repair time, procurement lead times etc. These data elements provide the life-cycle cost data and system data used for the ELORAs." To summarize, the computational model requires detailed information about the system utilization, component characteristics and maintenance site characteristics.

Questions for the above ELORA logic diagram

Question 1: Is the design of the item such that repair is feasible?

Question 2: Are the item's maintenance characteristics and installation such that a remove/replace strategy is feasible at first line?

Question 3: Does the item have lower level assemblies?

Question 4: Does item's maintenance characteristics permit a replacement action of the sub-item?

Question 5: Does the item's configuration consist of subassemblies from multiple vendors?

Figure 4: Example of non-economic LoRA questions [87]

An example of a LoRA model is the COMPASS (Computerized Optimization Model for Predicting and Analyzing Support Structures) tool developed by the U.S. army. [2] It runs in three modes:

1. Evaluate:

   The total maintenance policy cost is calculated using an economic model, given an end system and a defined maintenance strategy

2. Optimize:

   Minimizes total cost subject to constraints on the following factors

   - Economic factors
     - Cost
     - Availability
     - Mean time between failures (MTBF)
     - Mean time to repair (MTTR)
   - Non-economic factors
     - Safety
     - Mobility
     - Vulnerability
     - Policy
     - Manpower

3. STAT

   The Sensitivity and Trend Analysis Tool varies the values for the above factors to determine the effects of variability on the total cost

Essentially, COMPASS defines an optimization problem, minimizing cost subject to constraints defined by organizational rules and guidelines[3]. This is consistent

with the Navy's desire to "maximize the effectiveness of depot support and optimize program investments which will result in greater sustainment support to meet mission requirements"[113]

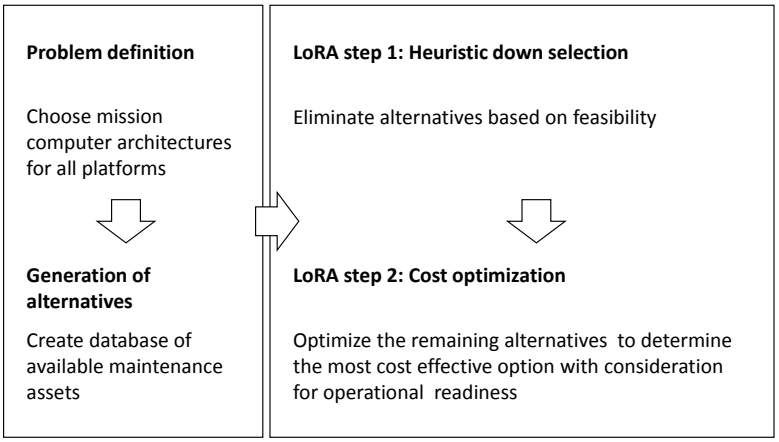


Figure 5: Overview of the Level of Repair Analysis methodology

1.3.3 Mathematical LoRA models

The computational LoRA models are primarily based on the methods described in the following literature. The problem is not particularly unique in its basic form, therefore the literature on the topic is fairly limited. To begin with, most of the work stems from the models of Barros [21][20] who poses the problem as an integer programming optimization problem.

Model assumptions:

1. A system is either operative or inoperative
2. A system's parameters are constant, such as the failure distribution
3. The operator knows the state of the system and failures are detected immediately

4. Component failures have no effect on other components in the system
5. Identification of the reason for failure is instantaneous
6. Failures follow a Poisson distribution with mean failure rate MTBF
7. If a subsystem is discarded so are all its components this is also true if it is shipped

Integer programming formulation:

- I - the set of indenture levels
- J - the set of components
- R - the set of available repair options
- Γ - a set hierarchical relationships between component set X
- $m(r,i)$ - 1 if repair option r is selected at indenture level i
- $n(r,x)$ - 1 if repair option r is selected for component x
- $c_v(r,x)$ - variable cost for using repair option r for component x
- $c_f(r,i)$ - fixed cost for enabling repair option r at indenture level i

$$\text{minimize} \quad c_t = \sum_{x \in X} \sum r \in R c_v(r,x) n(r,x) + \sum_{i \in I} \sum_{r \in R_1} c_f(r,i) m(r,i)$$

$$\text{subject to} \quad \sum_{r \in R} n(r,x) = 1, \text{ for } x \in X$$

$$m(r,i) \geq n(r,x), \text{ for } r \in R_1, i \in I, x \in X$$

$$n(r,x) \leq n(r,y), \text{ for } r \in R_2, \forall y \in \Gamma_x, \forall x \text{ non-terminal}$$

$$n(r,x) \geq n(r,y), \text{ for } r \in R_3, \forall y \in \Gamma_x, \forall x \text{ non-terminal}$$

$$m(r,i), n(r,x) \in 0,1, \text{ for } r \in R, i \in I, x \in X$$

The five constraints can be described as such

1. Exactly one repair option is chosen for each component

2. Fixed costs are charged if a repair option is chosen for a component

3. If a system is discarded so are its components

4. If a component is repaired so will its system

5. Restricts the problem to pure integer programming

Basten [22] generalizes this model by allowing a predefined set of components to share the same fixed costs. The following assumptions are made:

model assumptions:

1. Repairs have variable costs, annual fixed costs are incurred just to keep the capability of performing a given action at an echelon

2. The end item itself never moves for example the aircraft doesn't leave the operator; rather, the mission computers are replaced and move around

3. Repair a component by replacing a subcomponent, if it has no subcomponents modeled then it is repaired directly

4. A failed component can only be moved from an echelon level to e+1

5. Probability of successfully repairing is 100%

6. Data is aggregated across an echelon, meaning all echelons on the same level are considered as one big depot.

7. Following from the previous assumption, repair for a certain item always happens at the same site

8. Therefore, the number of locations between a site and the warehouse should be equal

At each echelon there are three possible decisions for each component. Either it is discarded or it is repaired. The third option is choose neither and send the component to the next echelon. Each LRU has an expected number of annual failures of which a percentage of these lead to SRU failures and so on. However, if a LRU is discarded then it doesn't matter what happened at the lower levels. Each decision, echelon, component triple is represented by a variable cost. Additionally fixed costs are associated with an echelon if any action is performed there for one component in a given set. This represents sets of components that utilize the same maintenance facilities and equipment, therefore as long as a location can service one of them, it can service all of them. This is the main contribution of Basten's model.

The following optimization problem is then stated using an objective function that is a sum of all variable and fixed costs for all components and echelons.

C_i : the set of components at indenture level i

Γ_c : the set of subcomponents of item c

g : a set of components that share a fixed cost

E : the set of echelon levels

D_e : the set of possible decisions available at echelon level e

$vc_{c,e,d}$: the variable cost associated with component c at echelon e to perform action d

$fc_{g,e,d}$: the fixed cost associated with component set G at echelon e to perform action d

γ_c : the total annual failures of component c

$X_{c,e,d}$: 1 if action d is performed for component c at echelon e

$Y_{g,e,d}$: 1 if for any component in G, action d is performed at echelon e

$$\text{minimize} \quad \sum_{c\in C}\sum_{e\in E}\sum_{d\in D} vc_{c,e,d}\cdot\gamma_c\cdot X_{c,e,d} + \sum_{g\in G}\sum_{e\in E}\sum_{d\in D} fc_{g,e,d}\cdot Y_{g,e,d}$$

$$\text{subject to} \quad \sum_{d\in D} X_{c,e,d} = 1, \forall c \in C_1$$

$$X_{c,e,move} \leq \sum_{d\in D_{e+1}} X_{c,e+1,d}, \forall c \in C, \forall e \in |e \neq e^{CEN}$$

$$X_{c,e,repair} \leq \sum_{d\in D_e} X_{b,e,d}, \forall c \in C|\Gamma_c \neq \emptyset, \forall b \in \Gamma_c, \forall e \in E$$

$$X_{c,e,d} \leq Y_{g,e,d}, \forall g \in G, \forall c \in g, \forall einE, \forall d \in D$$

$$X_{c,e,d}, Y_{g,e,d} \in \{0,1\}, \forall c \in C, \forall e \in E, \forall d \in D, \forall g \in G$$

The five constraints can be described as such.

1. There must be a decision for all LRUs at level 1. This is based on the assumption that the end item never leaves the operator

2. If move is chosen then a decision is made at the next echelon - since move is simply deferring the decision to the next level

3. If repair is chosen then a decision is made for all its child components at this level as they are now removed from the item and must be handled separately.

4. Fixed costs are taken into account if a decision is taken for the given component

Alternatively the LoRA problem can be modeled as a minimum cost flow problem with side constraints. A flow problem is a network representation of the echelons with $F_{v,w}$ is the amount of flow through a given arc and $Y_{r,l}$ indicates whether a given resource is located at a location. The network is constructed of source nodes representing the occurrence of failed components, decision nodes where variable costs are incurred as actions are taken, transformation nodes which represent the case where an item's components are separated and sent elsewhere, and finally sink nodes where

the flow goes when no more decisions need to be made. The constraints are such that outflow from source nodes is constant, decision nodes have balanced flow across them, transformation nodes split the inflow between the outgoing arcs, and only arcs that are enabled due to the availability of resources are used.

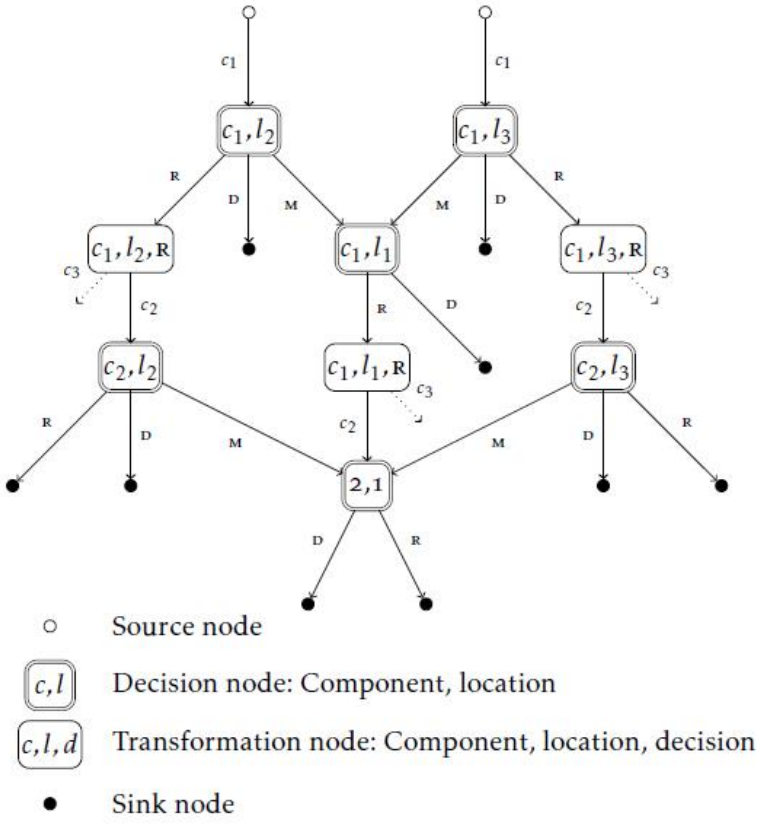


Figure 6: An example network flow problem ([22])

Bouachera et al. [31] suggests that a genetic algorithm would perform better than the traditional integer programming problem because of the large number of decision variables. It can be assume that gentic algorithms (GA) are good for complex, combinatorial problems.

Additionally tabu search is applied to the GA to improve the performance even further. In tabu search, [58] memory is used to keep track of solutions that have already been evaluated. In doing so, there is a better chance that the global optima is found.

To briefly summarize, LoRA is a critical part of the systems design process. It is an iterative process that is intended to provide different types of solutions at different times in the system's life cycle, from depot placement, maintenance equipment requirements, to spare component allocations. There are not many current methods for conducting LoRA, most of them use an integer programming model, they only consider the components for one system at a time, and only consider cost as an optimization objective. The Navy has stated that the maintenance strategy should ideally be low cost but it should also maximize the amount of time that the platform is available to the war-fighter, meaning that availability should also be considered. This would require a multi-attribute decision method.

Additionally, none of these methods consider the cost of changing the system once it is in place. It is assumed that when a change is needed LoRA will be conducted again and the appropriate change will be implemented. However, if it is assumed that changes are inevitable, such as in the HOST example, it is likely worth considering them from the start. Some potential disruptions that can be considered include required upgrades and part obsolescence. This is in contrast to the single platform analysis that assumes that upgrades and obsolescence do not effect the maintenance policy. In the past an upgrade of a system would simply require that LoRA be revisited and the appropriate changes made. However, in the HOST scenario multiple systems will be using a similar set of components and upgrading one system will likely have an effect on the other systems that share components. In this case which can be considered a multi-platform maintenance network, an upgrade to one platform system will effect the maintenance policies of all other connected systems. The same logic applies to the ability of the logistics network to reconfigure in order to support new platform types that will be added to the fleet and share components. Additionally the maintenance strategy should have the ability to transition to wartime levels of loading should the need arise.

The result is a modified maintenance planning problem. It is a multi-criteria optimization of a system subject to disruptions. In order to choose an appropriate method for solving this new design problem, the nature of the problem must first be defined.

1. Discrete alternatives

   Most traditional design methods rely on the fact that most of the design factors are continuous variables. This allows the use of highly efficient design space exploration methods such as gradient based optimizers, statistically informed designs of experiments, and time saving methods such as response surface modeling. However, for this type of problem, the design factors are the platform architectures, maintenance cite characteristics, component attributes, and operational requirements. The primary design variable in LoRA is the maintenance strategy for each component. This is generally characterized by a finite number of options which causes the design space to be discrete. While there are some continuous variables such as the labor rates of the depots or the failure rates of individual components, they are the exception instead of the rule.

2. Stochastic

   No two mission computers are truly identical, therefore they do not perform or fail the same way. While previous work tends to generalize and assume that all components fail a deterministic amount of times, the addition of discrete changes to the system due to disruptions invalidates this assumption. For example, upgrading a component will usually result in a slight change to its characteristics such as its failure rate. This discrete change in component attributes will cause a discrete change in the failure behavior. Additionally, it is very hard to predict when disruptions will occur, therefore these behaviors must be modeled as random events. This means that statistical analysis must

be done on each proposed solution. In order to enable the statistical analysis of each design, a significant sample set must be obtained using the evaluation method. This is done by repeating the evaluation of each design a number of times which will usually increase the computational cost by at least one order of magnitude.

3. Evolutionary development and emergent behavior

   "The Navy is called upon to continue to maintain weapons systems past their intended life while reconfiguring its depots to meet the maintenance needs of new systems designed for the evolution to the next generation of warfare".[155] As such the designed maintenance strategy will necessarily evolve over time as old platforms are retired or upgraded and new platforms are introduced. As mentioned several times previously, the effects of obsolescence on maintenance planning for modular systems should be significant. Therefore, the reaction of the system to disruptions must be considered.

4. Multi-objective

   "Maximize the effectiveness of depot support and optimize program investments which will result in greater sustainment support to meet mission requirements".[113] In addition to minimizing cost, the effectiveness of depot support must be broken down into several more metrics including, time to war-fighter and availability. Additionally, metrics must be placed on the system to quantify how well it evolves in response to disruptions.

To summarize, maintenance planning for the HOST problem is a multi-objective, stochastic design problem with discrete alternatives. It can be assumed that the maintenance network should never break down due to disruptions therefore, it is necessary to consider disruptions to the system and attempt to choose a design which

responds favorably in such an environment in addition to minimizing cost and maximizing platform availability. It will be shown in the next section that this design problem shares many characteristics with system of systems (SoS) design problems.

1.4 Systems of Systems

A system is defined to be "a functionally, physically, and/or behaviorally related group of regularly interacting or interdependent elements; that group of elements forming a unified whole".[117] From here it can be conceptually understood that an SoS is a group of complete systems that behave together in the way that subsystems are integrated to make a single system. However, here are some formal definitions that take into account additional factors.

1. Systems of systems are large scale integrated systems that are independently operable but are networked together for a common goal [88]

2. An SoS is defined as a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities [119]

3. Systems of systems exist when there is a presence of a majority of the following five characteristics: operational and managerial independence, geographic distribution, emergent, behavior, and evolutionary development.[134]

4. Systems of systems are large scale concurrent and distributed systems that are comprised of complex systems [95]

5. An SoS involves the integration of multiple, potentially previously independent, systems into a higher level system [146]

In summary, the following characteristics are crucial to understanding any kind of SoS:

1. Independent systems

   The systems that make up the SoS, are independently owned and managed. This also describes how the individual systems have frequently been designed at different points in the SoS life cycle or are purchased COTS products and were not necessarily designed to work well together.

2. Networks and interoperability

   As mentioned above, the independent systems must work together. In order to do that, they must share resources, and a network is a graphical representation of all the resource sharing links in an SoS. The interoperability is a measure of how well those links are implemented. Understanding how the systems collaborate and share resources is crucial to quantifying SoS performance.

3. Emergent behavior

   The phenomenon of emergence will be defined as the fact that the performance of the SoS is not equal to the sum of performances of the individual systems. In other words, the SoS's ability to share resources can help fill in for the deficiencies of some individual. Alternatively, the requirement to share resources can also encumber the SoS and reduce the overall performance. Only proper analysis can determine which behavior will occur.

4. Evolutionary development

   This describes how the SoS is intended to adapt to environments that are outside of its originally intended operational envelope. Given that the cost of implementing an SoS includes the cost of many potentially expensive systems, an SoS is usually intended to outlive the original scenario it was designed for. Additionally, it is frequently a requirement that the SoS never fail. As such, less than optimal performance is acceptable for short periods of time as long as

the capability continues to be available. Therefore, the ability for the SoS to adapt to disruptions and changes is crucial for a successful SoS.

To bring it back to the multi-platform logistics problem, all of these characteristics were described in the previous section. For the most part the different platforms and their maintenance supply chains can be considered independent systems. They were likely designed at different times for different purposes and with different maintenance requirements. However, now that a set of common components is introduced, those previously independent supply chains must now overlap and share resources. Emergent behavior can be observed as the resource sharing allows for the system to get away with stocking fewer spare components and potentially reduce the cost and time required to implement an upgrade. Finally, as mentioned in the previous section, disruptions must be considered forcing the system to adapt and evolve in response.

In order to further narrow down the definition for the purposes of this study, it is necessary to further differentiate between types of SoS.[119]

Virtual SoS lack a central management authority and a centrally agreed upon purpose for the system-of-systems. Large-scale behavior emerges and may be desirable, but this type of SoS must rely upon relatively invisible mechanisms to maintain it

In **collaborative** SoS the component systems interact mostly voluntarily to fulfill agreed upon central purposes. The Internet is a collaborative system. The Internet Engineering Task Force works out standards but has no power to enforce them. The central players collectively decide how to provide or deny service, thereby providing some means of enforcing and maintaining standards.

Acknowledged SoS have recognized objectives, a designated manager, and resources for the SoS; however, the constituent systems retain their independent ownership, objectives, funding, and development and sustainment approaches.

Changes in the systems are based on collaboration between the SoS and the system.

Directed SoS are those in which the integrated system-of-systems is built and managed to fulfill specific purposes. It is centrally managed during long-term operation to continue to fulfill those purposes as well as any new ones the system owners might wish to address. The component systems maintain an ability to operate independently, but their normal operational mode is subordinated to the central managed purpose.

Most SoS are virtual, some are becoming collaborative, and even now there are the beginnings of directed SoS that are designed from the ground up. However, the DoD puts its focus somewhere in the middle as it sees a growing number of acknowledged SoS where the individual systems are independently developed but expected to work together. This is the case with the maintenance planning problem.

The additional defining characteristics of acknowledged systems of systems are as follows.

Provides a unique capability or common goal

A capability is "the ability to execute a specified course of action (A capability may or may not be accompanied by an intention.)." [90] Unlike a virtual SoS, an acknowledged SoS requires that all the entities have agreed upon a common goal, that they are working together to perform a single high level task. In this case the common goal would be to provide maintenance support for all the naval aircraft.

Geographic distribution

While not a truly necessary condition for an SoS it is a common issue. The consequences of physical distance between systems include time delays on resource transfers and tend to reduced the efficiency of the SoS. For this problem

the maintenance sites can be thousands of miles apart resulting in significant shipping times and costs.

1.5 SoS design

Given that the multi-platform maintenance planning problem has been characterized as an SoS design problem, it is prudent to explore the literature on SoS design methodologies in order to find one that may work for this problem. The first place to start is with the history of systems design methodologies. The evolution of systems of systems design methodologies begins with methodologies originally formulated for large scale software development.

The first of such development models was the waterfall model. it was developed by Winston Royce in 1970 [131] to address the fact that the traditional software development process of analyzing the problem then writing code was unable to handle the development of larger, more complex systems. The model introduces a set of defined steps that are followed in order with a minimum of iteration necessary. The first step is to define the system and its requirements. Then the problem is analyzed and a design is proposed. If no design successfully meets the requirements the model allows for iteration on the requirements until a design can be found. The next step is to implement the design and test it. If after testing, the design it is determined that it is not good enough, the model returns to the design step and iterates until a good design is found and is implemented. (Figure 7)

A second method was developed by J Harvey Evans in 1950 [59]. The spiral model was intended to address the fact that for complex systems such as aircraft and naval vessels, the increase in number of requirements makes the effects of early decisions more pronounced. The model organizes the required design and development tasks into a logical order and as the development effort spirals out the early decisions are revisited and verified and a better solution is converged upon. (Figure 8)

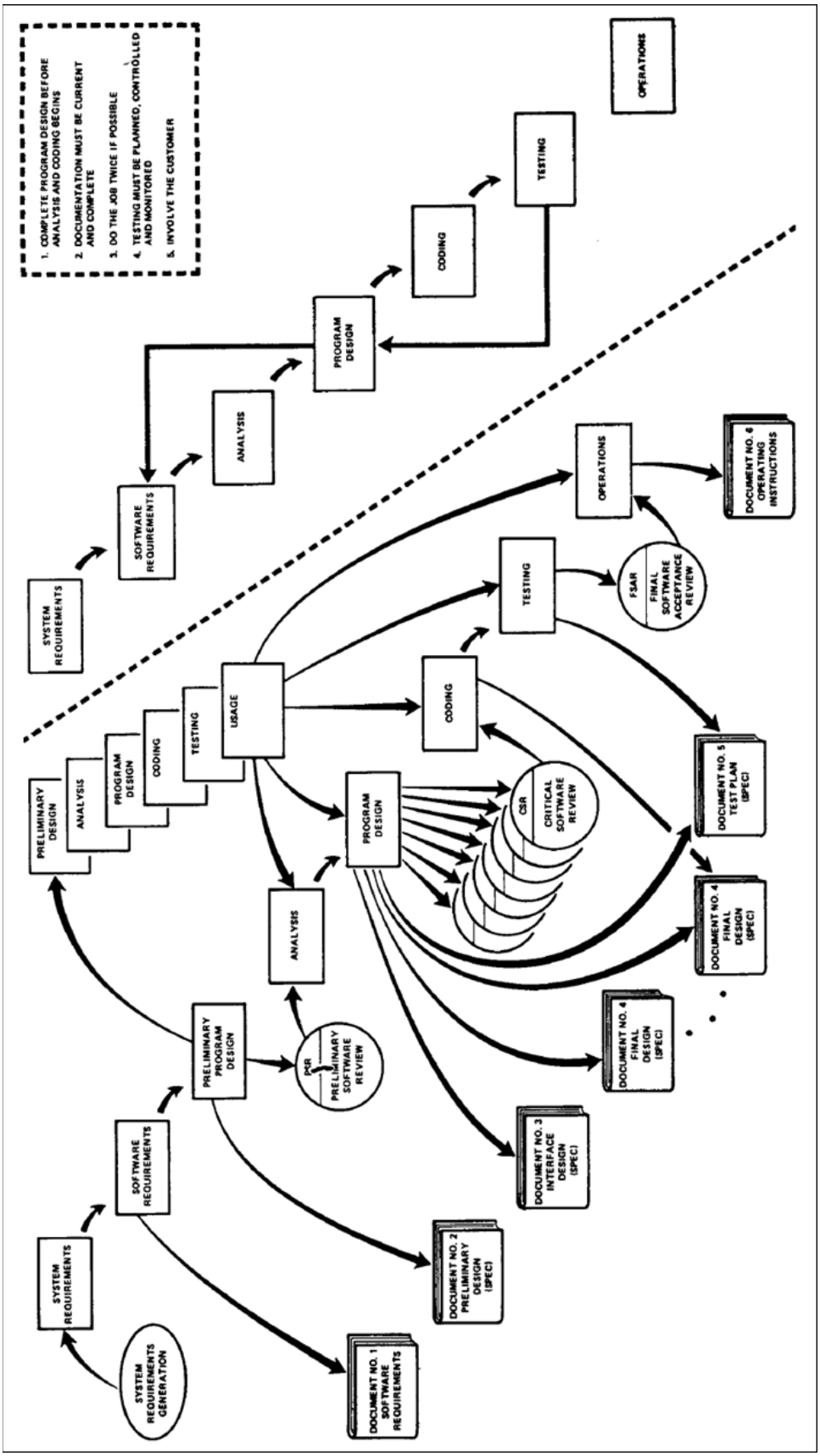


Figure 7: Waterfall model for software development [131]

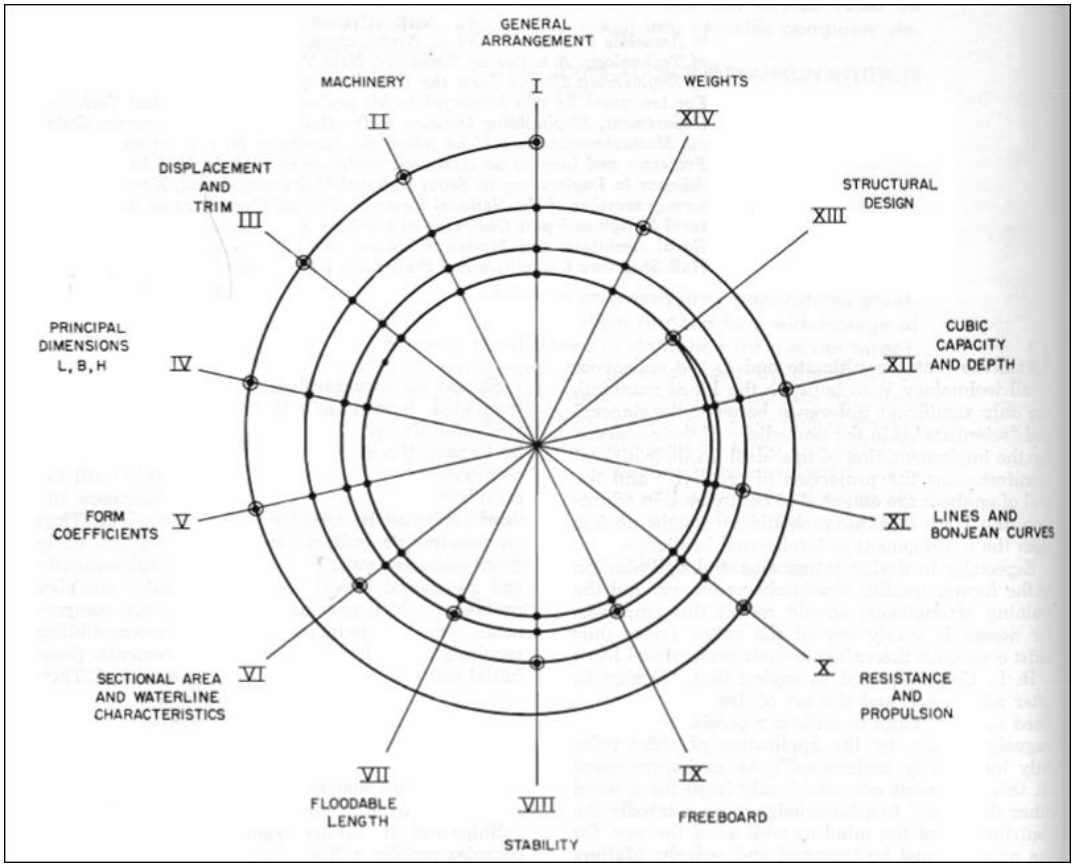


Figure 8: Evan's spiral model for ship design [59]

Boehm modified this method in 1988 [28] (Figure 9) to apply to software development efforts and breaks the spiral into four phases.

1. Determine objectives, alternatives and constraints.

2. Evaluate alternatives, identify and resolve risks.

3. Develop and verify product

4. Plan the next phase

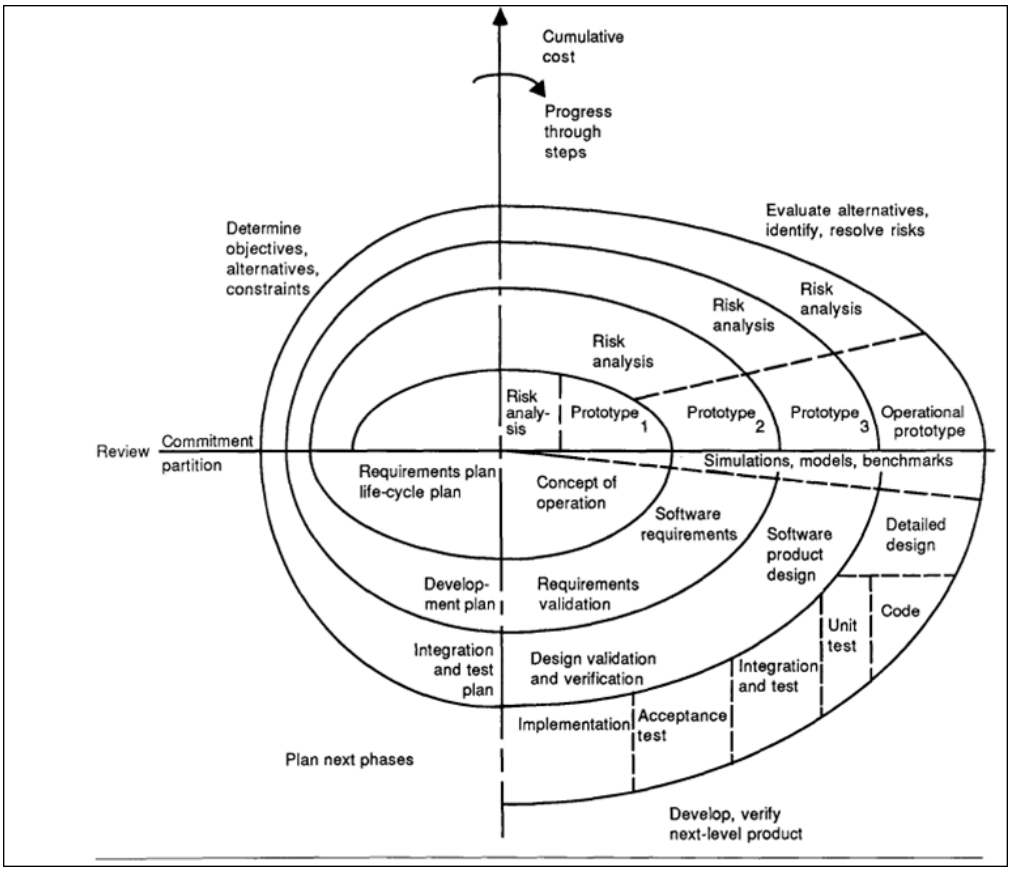


Figure 9: Boehm's spiral model for software development [28]

The next generation of design models was the vee model. Two versions of the vee model were developed almost concurrently. The first by NASA in 1987 [112] for software development and the second by Forsberg and Mooz in 1990 [71]. Both models

are essentially a series of waterfall models linked together to handle the design of very complex systems. Each step on the vee corresponds to an increase in granularity in focus. For example, while the first step may be designing the general functionality of the system, the later steps will be the design of increasingly more detailed components. While the left side of the vee represents this decomposition and design of the system and its components, the right side is a decrease in detail as the components are implemented and tested finally culminating in the implementation of the entire system.(Figure 10) The ability to consider increasing levels of abstraction makes this method attractive for the design of systems of systems simply by adding one more level to the vee. However, this model does not consider life cycle effects or the fact that emergent behavior makes the link between decomposition and recomposition tenuous at best.

To summarize, this model begins to show some generic methodological steps that can be used for SoS design. The first step must be to define the problem and determine the objectives and the constraints. Then the different alternative designs can be evaluated and the best ones can be implemented. These steps will be the primary backbone of any design methodology.

Moving on to design methods specific to systems of systems, SoSE is the process of designing an SoS. It is defined by the SoSE center of excellence as "an emerging interdisciplinary approach focusing on the effort required to transform capabilities into SoS solutions and shape the requirements for systems. SoS Engineering ensures that: Individually developed, managed, and operated systems function as autonomous constituents of one or more systems of systems and provide appropriate functional capabilities to each of those systems of systems, Political, financial, legal, technical, social, operational, and organizational factors, including the stakeholders' perspectives and relationships, are considered in SoS development, management, and operations, An SoS can accommodate changes to its conceptual, functional, physical,

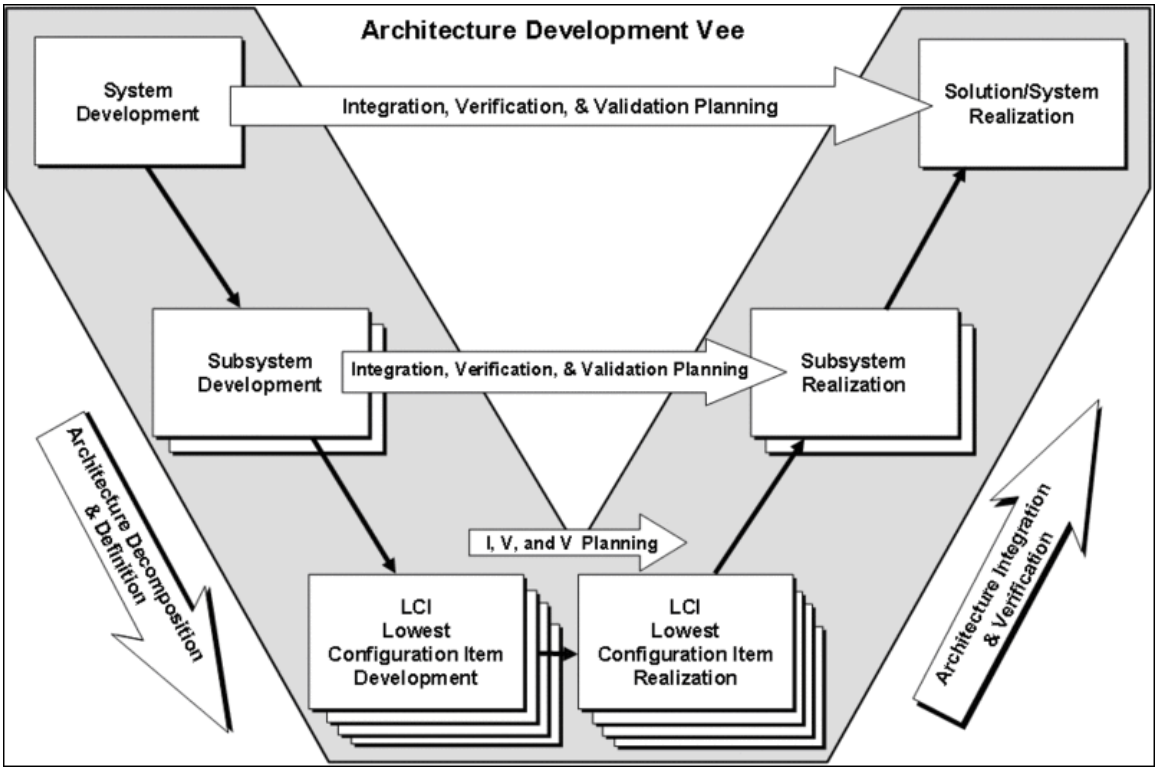


Figure 10: Systems engineering vee model [72]

and temporal boundaries without negative impacts on its management and operations: An SoS collective behavior and its dynamic interactions with its environment to adapt and respond, enables the SoS to meet or exceed the required capability." [149]The literature on the subject prescribes a number of methodologies and design philosophies for approaching SoSE, including the following.

Keating defines SoSE simply as the integration of complex systems. He identifies the main challenges of SoSE as the uncertain nature of the design requirements, the fact that SoS performance is dependent on the conditions that the SoS operates under, the human element, and organizational and political issues that tend to plague projects on such a large scale. Due to the ambiguity of the final design, it is very difficult to determine the boundaries of the design space as is necessary for the "problem definition" step of conventional SE methods. Therefore, the keys to SoSE success lie in a structured, iterative, multidisciplinary process for designing a robust and flexible solution. Also known as a "satisficing" solution. In essence the focus must shift from designing final products to picking good enough initial deployment options that will adapt and evolve over time. [146]

Luzeaux adopts a similar approach, pointing out that the goal of SoS design is not optimal performance but satisfactory performance. An SoS is usually comprised of large heterogeneous and autonomous systems, defined by a focus on networks and resource sharing, and largely dependent on human behavior. Therefore, it is impossible to truly quantify the performance of the SoS during the design process. Only, later on through observation and feedback is the SoS capable of performing its task and performance can be quantified. As such the focus must be shifted from the individual specifications of the component systems to their interactions. [100]

Alternatively Sage and Cuppan identify the organizational issues to be the biggest roadblock to SoS design. They propose that to successfully implement an SoS it must be governed as a whole, hence they call it a federation of systems. In order to realize

this goal, they call on the tenets of new federalism to prescribe how the design teams and governing organization should be arranged and how they interact. The result is an environment that fosters innovation which is crucial to survival in an evolving environment as long as it adheres to the common goal. [134]

To summarize, the focus of SoS design is to use efficient analysis methods that may be lacking in accuracy to rapidly implement a design that can perform the capability adequately. Once it is up and running the focus shifts to ensuring that it continues performing its job despite changes to the environment. This is achieved through continuous observation, feedback and proper management practices that promote adaptation. To use an analogy, where traditional systems engineering is focused on developing an airplane and getting it in the sky, SoS engineering is focused on changing the airplanes engines mid-flight. Therefore, since the SoS can never be taken down for maintenance, the early steps of SoS design must focus on generating designs that are easy to adapt using metrics such as robustness and flexibility. In essence, it is like designing an airplane where it is easy to change its engines mid-flight.

On this note, the DoD developed its own model for SoS development (figure 11) focusing on the needs of the individual systems and acknowledging that the process must continue throughout the life cycle of the SoS, since systems of systems are not usually new acquisitions, rather they tend to be an evolution of the interactions between a group of existing systems. [119]

The major criticism of this methodology is that the order of steps is fairly ambiguous and there doesn't seem to be a well defined output from the methodology which leads to its metaphorical nickname, "The taxpayers' sinkhole of SoS development". Therefore, for the implementer's ease Dahmann modified the DoD's model to represent it from a sequential time perspective, specifying where feedback occurs and clarifying where the SoS actually gets implemented. [47] (figure 12)

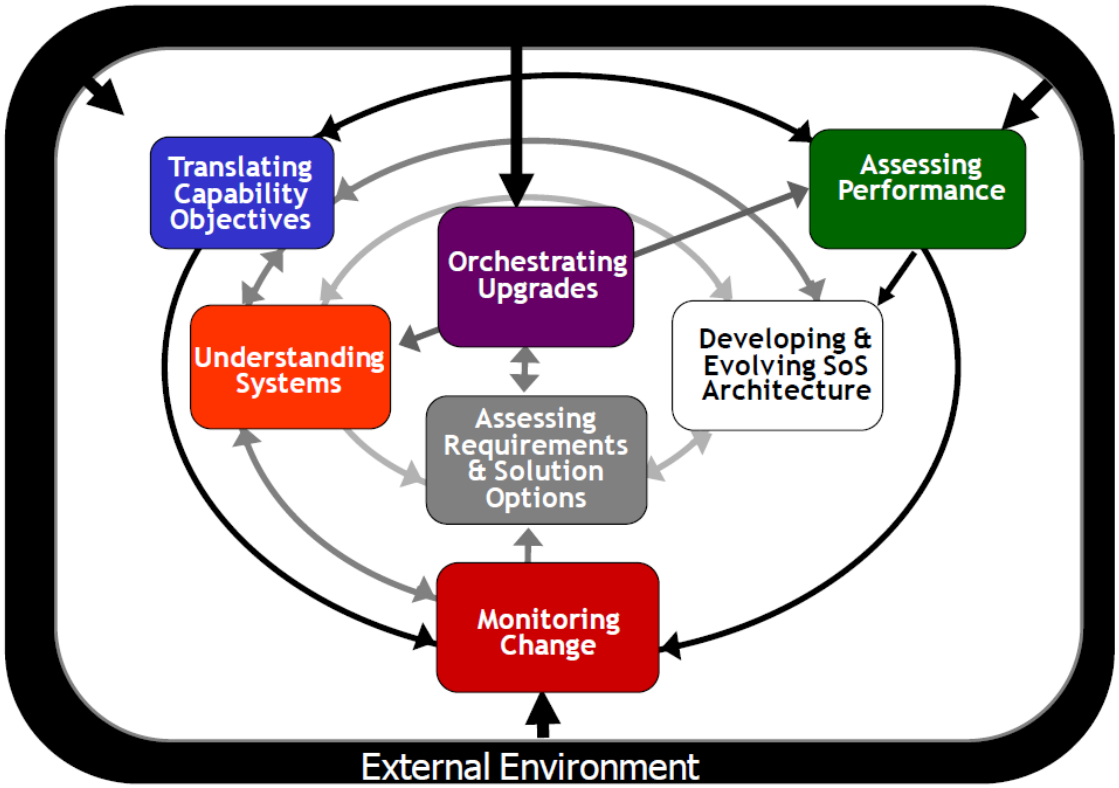


Figure 11: DoD trapeze model [119]

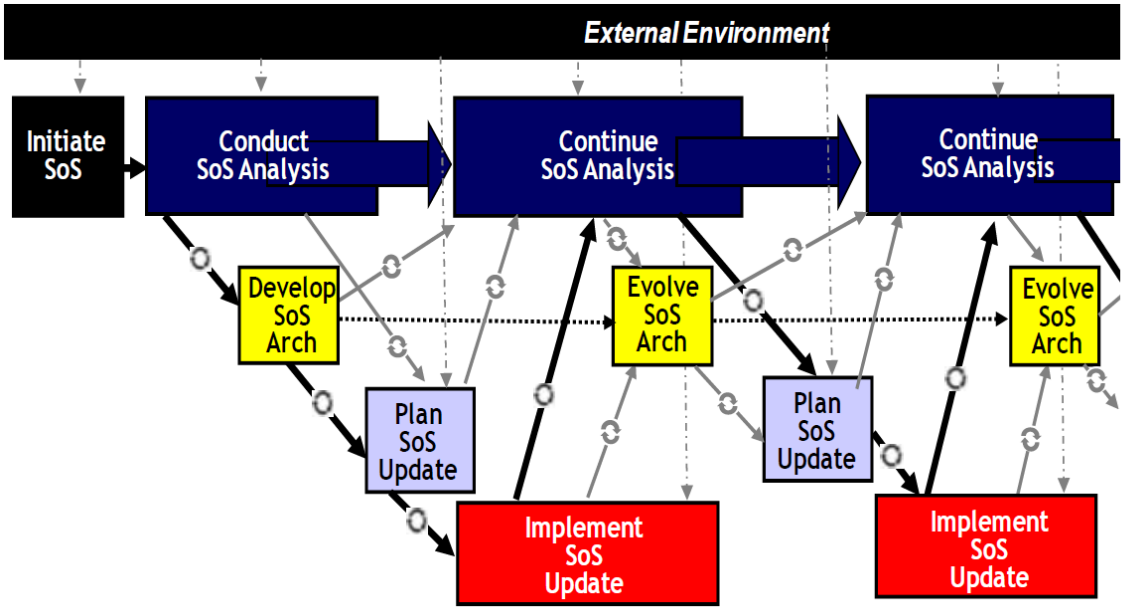


Figure 12: Dahmann's Wave model [47]

Similar to the DoD, Delaurentis defines an outline for an SoS design process focusing on what must be done before decisions can be made. The first step is the definition phase, where the baseline SoS is set and barriers to more preferred behavior are identified. Next is the abstraction phase, where the main actors, resources, disruptors, drivers and networks are defined. Finally with all that in hand, modeling and simulation takes place in the implementation phase, at which point the decision making process can begin.[51] This method does not consider the iterative nature of SoS implementation, though it does give a good sense of how the first steps of SoS development should occur.

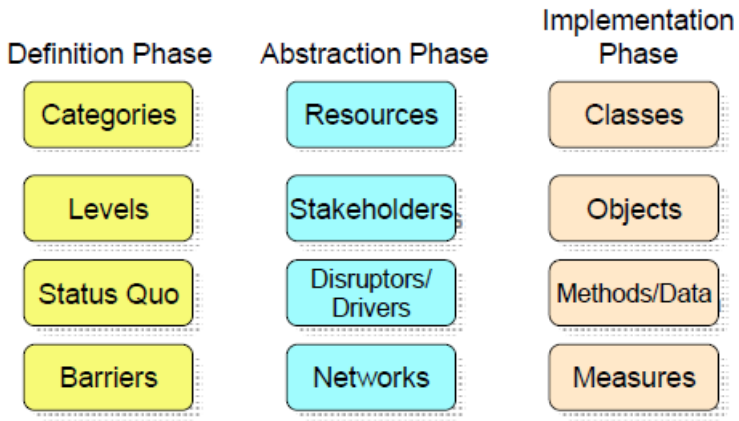


Figure 13: Delaurentis' "Proto-method" for SoS problems [51]

Most recently, Griendling modified the traditional systems engineering V-model to facilitate capability based acquisitions. The focus was on how the SoS is represented and documented during the design process using architectural views. The

methodology defines how architectures can be used for alternative generation and evaluation. The result is a streamlined process for designing a single update to an SoS with consideration for supporting the Wave model throughout the life cycle of the SoS [77]

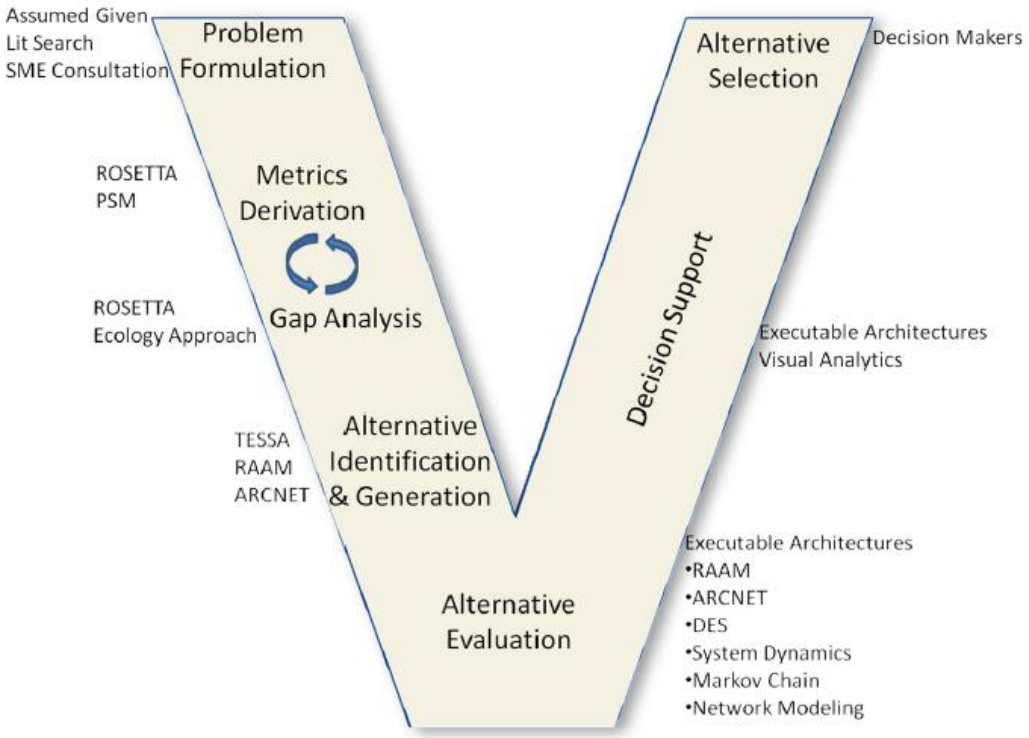


Figure 14: ARCHITECT Vee with Enablers Mapped to Methodology Steps [77]

To summarize, the first two models (DoD and Wave) focus primarily on the iterative nature of SoS implementation. They were developed for the purpose of continually supporting the SoS while it is in operations through feedback and upgrades. However, the problem with them is that they are very abstract and hard to follow. The second two models (Delaurentis and Griendling) focus on a single iteration of the SoS design cycle. The propose methods for problem definition, alternative generation and design evaluation in support of the decision making process. However, neither method does much to account for the ability of the SoS to adapt and evolve.

The goal of this thesis is to iterate on the above design methodologies and define a methodology that will work for the multi-platform maintenance planning problem. First, it is clear that for SoS design problems it is necessary to consider how the SoS will evolve over the course of its life. This includes planning for upgrades even if the exact content of those upgrades is impossible to predict due to the uncertainty in the future operating environment. Whether it is providing for a mechanism to iterate on the design or planning on observing the SoS and upgrading it as needed, all the design philosophies described here attempt to pick an initial design that can be easily modified in the future. This is somewhat contrary to the systems engineering philosophy that early decisions matter greatly and therefore great care must be taken in making those decisions. In fact, the only decision that must be made early on for systems of systems is how to delay the decisions as long as possible by implementing a system that is easy to adapt over time.

Figure 15 shows the beginning of the development of a methodology. It starts with a problem definition phase where the boundaries of the system being designed are defined. Then objectives of the design process are stated, such as minimizing O&S costs and maximizing performance. Finally, potential disruptive conditions must be listed, which is similar to the barriers and disruptors of favorable behavior described by DeLaurentis. The second step is the modeling process where computational methods are used to estimate the goodness of potential designs with respect to the design objectives. This is where many of the contributions of this thesis will fall given that modeling and simulation is one of the biggest gaps in SoS design. However, the nature of SoS modeling is such that it is not likely that all possible designs will get evaluated at the maximum level of fidelity. Therefore, the third step will be a down selection of the design space, so that a working solution can be implemented as quickly as possible. As such, the transition between steps 2 and 3 is iterative, where the design space is systematically reduced using increasingly more reliable models, before advancing

to the final step. In step 4, decisions are made and the chosen system architecture is implemented. This is done by using knowledge of the current scenario in order to tradeoff between the different objectives and pick the design that is best suited to the given customer's priorities. Finally, since the design of an SoS is never complete, the methodology repeats itself by observing the system in operation and reevaluating at some later point in time when changes need to be made.

As it stands, this methodology is incomplete. Therefore the purpose of this research is to make an attempt at filling those gaps in the context of the multi-platform maintenance planning problem. The first question is, how can the system be evaluated in the context of a disruptive environment? This will primarily be answered by developing recommendations on metrics to use and how to model them. The second question is, given that SoS modeling is expensive and the design spaces are large, are there ways of reducing the cost? This leads to the third question. Given that it may not be possible to improve the modeling costs by enough due to the complexity of systems of systems, are there efficient methods of down selecting the number of designs so that the higher fidelity models only get used on the most promising designs? The following sections will attempt to answer these questions.

1.6 Flexibility measurement

1.6.1 Definitions of flexibility

The first question that must be answered is how to quantify the ability of the SoS to continue operating despite disruptive events. The literature in the previous section implied that a flexible system would be able to adapt to changes in the environment and maximize the chances that capability is performed. The following section will be a literature review of the definitions and methods for measuring flexibility.

Several engineering fields have attempted to design flexibility into their systems, including manufacturing, space, communications, and transportation systems. Each

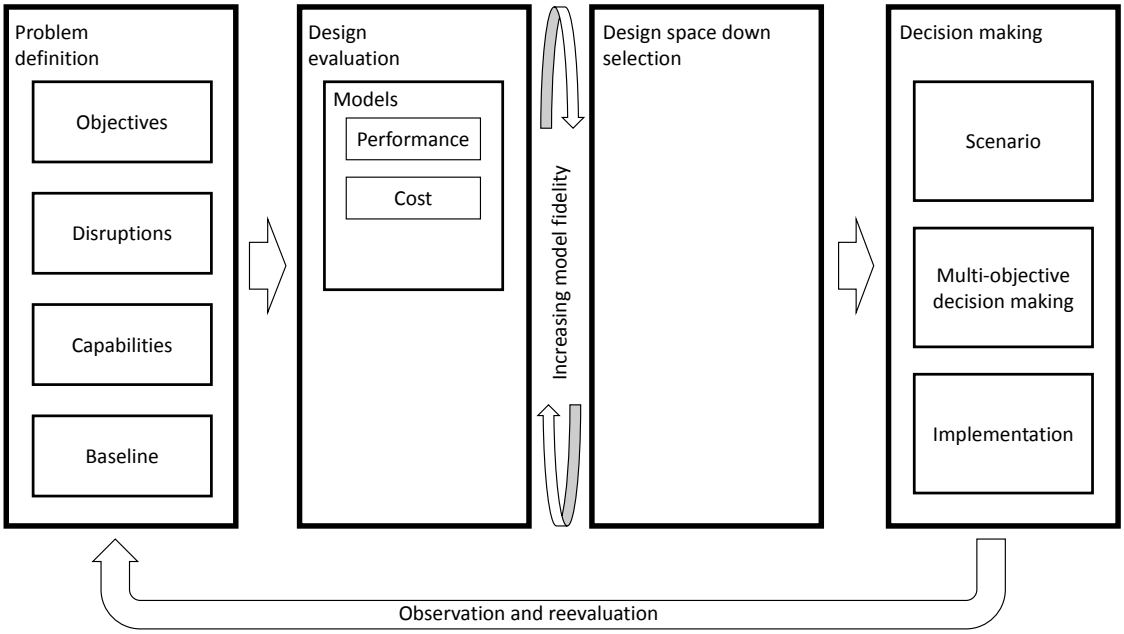


Figure 15: Initial skeleton of the proposed SoS design methodology

of these design problems desire flexibility to mitigate the effects of disruptive events that could cripple the system. In some cases it is because the system is hard to access and fix, in other cases it is because system failure would be catastrophic, but in all cases flexibility is desired because of the massive amounts of money invested in the system.

From the literature, three categories of definitions can be defined. The first set focuses on the ability of the system to adapt to changes. If a model can be defined for how the system will adapt to disruptions then these definitions are likely to be sufficient. However, in general, it is impossible to accurately predict all the disruptions and therefore it is equally difficult to predict how it will adapt. The following are some of those definitions from the literature.

- A measure of the ability of a system to adapt to external changes, while maintaining satisfactory system performance. [109]
- The ability to respond to a changing environment [24]

- The ability of a system to respond to potential internal or external changes affecting its value delivery, in a timely and cost-effective manner. [115]
- The ability to change or react with little penalty in time, effort, cost, or performance [153]
- The ability to adapt to changing environment (technology) and requirements (market demand) [165]
- The ability of a system to respond to changes to its mission in a timely and efficient manner [135]
- The ease with which a network can adjust to changing circumstances and demands [62]
- The capacity to continue functioning effectively despite changes in the environment [103]
- The ability of an organization to suffer limited change without severe disorganization [61]
- The extent to which the structure of an organization facilitates or hinders responsiveness of members of the organization to change [129]
- The ability to adjust activities to changing conditions [97]
- The three categories of flexibility for systems requiring manned labor are, numerical flexibility, functional flexibility and financial flexibility. Numerical flexibility is the ease by which the number of employees can be changed to meet changes in demand. Functional flexibility is the ease by which the labor tasks are changed, and financial flexibility refers to how the payment structure enables the other two categories. [13]

Many of these definitions emphasizes the ease of adapting effectively. Some focus on the cost of adapting, some focus on the time required to implement the change and others focus on the magnitude of the loss of performance due to a disruption. In practice the definition that gets used is likely going to correspond to the metric of primary importance to the problem. In traditional maintenance planning that metric would be cost, however for systems of systems maintaining an adequate level of performance is the primary goal, therefore quantifying the loss in performance would make sense.

The second category of definitions focus on the range of capabilities or decisions that the system possesses. In theory a system that has more options will be more resistant to change as it can pick whichever option maximizes performance given the current environment. For example, if an intelligence, reconnaissance and surveillance (ISR) system can choose from a satellite, a low flying aircraft or a special operations commando to track a target then it will be able to switch off between modes of sensing depending on whether there is cloud cover, or the target goes into a densely populated area. Similarly, the more options the system has for performing its job then the less likely an event that disrupts some of the options will entirely cripple it. For example, if a maintenance system has two available depots then if one is shut down it will still be able to perform repairs. In essence, this set of definitions comes very close to the concept of resilience which can be defined as the system's ability to sustain damage. [63] The following are some of those definitions from the literature.

- The number of alternatives left over after making a decision [130]
- The postponement of decisions until more is known [83]
- The capacity for taking new action to meet new circumstances [103]
- The cost of moving between states, the loss of performance when changing states, and the physical difference between two states [48]

- The range of reachable states, and cost to transition [143]
- A measure of the system's potential performance [142]

This set of definitions tends to quantify flexibility as the absolute number of options available to the system. For example, a system that can perform multiple tasks will still be valuable as long as at least one of those task is required, therefore it can be considered more flexible than the system that only has one purpose. As before, some of the definitions also identify the cost of accommodating all these possibilities as a primary concern. To use the maintenance system above as an example again, sustaining two depots is likely to cost more than only having one.

The final category of definitions focuses on the system's sensitivity to uncertainty. In essence, these definitions mirror the concept of robustness which can be defined as the variability in the systems behavior with respect to environmental noise. [107] In theory, a system that is invariant with respect to disruptions could also be considered flexible in the sense that it will continue performing its job despite variations in the environment. However, in reality the two concepts tend to be opposites. Robustness tends to describe how the system doesn't need to adapt most of the time and does not consider what should happen when the system is finally outside its comfort zone and is asked to adapt. Given that for systems of systems, the true nature of the disruptions is impossible to predict, it is necessary to consider the ability to adapt. However, robustness should be a component of SoS flexibility since a robust system will be able to maximize the time required between upgrades. The following are some of those definitions from the literature.

- A filter for external perturbations, an absorber of uncertainty [45]
- Excess resources in the system for dealing with uncertainty [104]
- A hedge against uncertainty [64]

To summarize, the definitions fall into the following three categories.

1. The ability of the system to adapt to change
2. The number of options available to the system
3. The systems sensitivity to uncertainty

These categories are somewhat related. For example, if a system has the capability to perform many tasks, then the average cost of adapting to a change in requirements will be low since some of the time it will already possess the required capability and the cost will be minimal. Another example, if a system is very good at adapting then it is more likely to succeed in an uncertain environment, because it is less likely to cost as much to change to match the true nature of the requirements once more information becomes available. These three categories will form the basis for a generic definition of SoS flexibility. While it is tempting to choose one definition and run with it, it is better to consider flexibility as an aggregate metric that combines the effects of adaptability, robustness and resilience into one system attribute. Flexibility can be summarized as the ability of the system to continue performing its job in a disruptive and evolving environment. It is a quantification of the evolutionary nature of the system though that is too abstract to actually measure, therefore a multi-faceted definition is easier to implement.

Another product of this literature review is an initial indication of how different aspects of flexibility are measured. What will become more apparent in the next few sections is how problem specific the actual measurement methods tend to be. In those sections the literature will be surveyed further in an attempt to find methods for measuring flexibility that apply to the multi-platform maintenance planning problem and systems of systems in general. Below is a summary of five different measurable quantities that were linked to flexibility by the sources mentioned above.

1. Cost of changing

2. Time to effect a change

3. Loss in performance due to a change

4. Number of available options

5. Amount of surplus resources

1.6.2 Flexible manufacturing systems

Much of the literature on measuring flexibility originates with manufacturing systems. When developing a new product, much thought is given to the design of the manufacturing line that will produce the item. This initial factory setup represents a major initial investment in the product. However, should circumstances change, it might be necessary to make changes to the manufacturing system which would represent another major investment. For example, should the demand for the product increase, failing to increase the capacity would result in a lost opportunity. Alternatively, if a new product is developed, it would be advantageous to utilize as much of the existing manufacturing system as possible in order to minimize the initial setup cost. Therefore, much thought has been put into how to design flexible manufacturing systems. To do that it is necessary to measure the value gained by being flexible.

Most of the measures focus on a number of different categories of flexibility in the system, each one corresponding to either certain types of changes that are expected to occur in order to minimize the impact or typical actions that must be performed when attempting a change in order to minimize the associated costs. They are:

- Slack [142, 143, 144] identifies seven categories of flexibility based on the range of reachable states and the cost to transition. The categories are machine, routing, process, new product, delivery, mix and volume.

- Browne [34] identifies a slightly different set of categories as machine, product, process, operation, routing, volume, expansion,and production flexibility.
- Zahran [164] focuses on he number of alternatives for processing each product, and the efficiency in changing to process new products.
- Kochikar [93] focuses on the cost of changing states measured with a reachability graph of all available states.
- Shewchuk [140, 141] develops generic measures for product, routing, material, production, volume, and mix flexibilities.

Sethi and Sethi [137] summarize the list of categories of manufacturing system flexibility and defines them as such.

- Machine flexibility relates to the equipment that is used on the manufacturing line. It is a measure of the number of operations that each machine can perform without a significant effort required to switch.
- Flexibility of a material handling system is a measure of the system's ability to move different part types efficiently for proper positioning and processing through the manufacturing facility.
- Operation flexibility is the number of ways to produce a certain part.
- Process flexibility is a measure of the set of part types that the system can produce without major setups.
- Product flexibility is the ease with which the system can change to accommodate new parts.
- Routing flexibility is the number of alternate routes through the system for producing different parts.

- Volume flexibility is the range of demand levels that the system can operate profitably at or, given a distribution of the demand, the probability that the system is profitable.

- Expansion flexibility is the ease with which the system's capacity and capability can be increased when needed.

- Program flexibility is the ability of the system to run virtually unattended for a period of time.

- Production flexibility is the maximum number of part types that the manufacturing system can produce without adding major capital equipment. This is also sometimes referred to as mix flexibility.

- Market flexibility is the ease with which the manufacturing system can adapt to a changing market environment.

These measures very closely follow the categories listed previously. When the ease of adaptation is measured it is generally done by quantifying the cost and time required to effect that change. When the number of available options is desired, it is simply counted. Finally, resistance to uncertainty generally refers to the range of environments in which the system can operate comfortably. As the environment that a manufacturing system exists in is usually defined by the demand for goods, this is usually measured by the probability that the system is profitable given a range of demands. Sometimes the amount of excess resources that are available to accommodate increases in demand is used as an indicator of this resistance. In theory if the manufacturing line has some unused machines lying around, then when the demand increases those machines will go towards filling the new demand.

Later work has been devoted to exploring the relationships between these categories of flexibility and the overall effects on performance and cost.

Shewchuck [141] explores tradeoffs between product, mix, production and volume flexibility. The results indicate that some types of flexibility can be increase mutually such that trade offs are not always present if the correct design approach is chosen. One result was that product flexibility increases with part flexibility. This usually also comes with an increase in the processing capabilities of the system and the processing capabilities of individual groups of machines. Additionally, mix flexibility increases by employing a larger quantity of less flexible processors if processor flexibility is low. Otherwise increasing the processing capabilities of the system has the same effect. Finally, production flexibility increases with the processing capability of the system and volume flexibility increases with the processing capacity of the system.

Chen and Chung [42] explore the relationship between machine flexibility, routing flexibility, and system performance. Performance was quantified by machine utilization time and the number of in process items. The result is that flexibility generally increases performance at a decreasing rate. Routing policy as well as operating conditions could have critical effects on the magnitude of performance improvement.

Benjaafar [25] investigates the correlation between production and machine flexibilities with performance. One result is that flexibility can be an effective mechanism in dealing with the disruptive potential of variability in a dynamic environment. Additionally, in the presence of variability a relationship can be shown between flexibility and performance. This is shown by a reduction in part flow time, number of bottlenecks, part waiting time, and work-in-process inventories. Another conclusion is that flexibility has an effect on average performance and the variance on performance. More flexibility results in better performance and more predictability. These benefits can be found at even low levels of flexibility but there are diminishing returns on how much performance can be obtained.

Gupta et al. [79] explore the trade offs between multiple types of flexibility and performance. Performance is quantified by the machine idle time which is the inverse

of machine utilization and job waiting time. The first result was that job waiting time increases with product, machine, and process flex which are all mutually related. This also indicates a trade-off against volume flexibility. In general, adding more machines was found to tip the scale in favor of product flexibility. The second result was that machine idle time decreases when the number of machines and routes is simultaneously decreased. This causes an increases in volume flexibility and product variety.

Suarez et al. [147] explore the relationship between product flexibility, mix flexibility, volume flexibility and performance. Unlike, all the previous studies, this study used data collected from existing manufacturing plants. The findings were as follows

- More automated plants tended to be less flexible despite the fact that the machines are more reconfigurable
- Work involvement and flexible wages correlate with an increase in flexibility
- Component re-usability increases mix and product flexibility
- There is no apparent relationship between product quality or cost and flexibility
- There is no apparent relationship between mix and product flexibility
- Volume fluctuation and mix flexibility are negatively correlated. This is because the greater the mix of products the less likely a change in demand for one product type will greatly affect the system.
- Volume and mix flexibility showed no relationship. However, in theory a system with low mix flexibility will need high volume flexibility to compensate.
- There is a weak positive correlation between volume and product flexibility.

- Mix and product flexibility are strongly correlated. The makes sense since the faster a plant can introduce new products the more likely it will be to increase its mix.

To summarize, there are trade-offs between different types of flexibility, but in general any increase in flexibility will result in an increase in performance in the presence of disruptions. In conclusion, there are many useful concepts that can be borrowed from manufacturing system flexibility, but the actual measures tend to be limited in scope as they are very problem specific as they frequently relate to the physical equipment that performs the tasks. However, one concept that seems promising is the idea of volume flexibility which is a quantification of the range of demands that the system can accommodate. In the maintenance planning problem, the demand on the system can be defined in terms of the rate at which components fail and need maintenance, therefore a measure of the maximum failure rate that the system can effectively handle would be a meaningful measure. To generalize this to other types of systems of systems it would be meaningful to know the largest enemy that a military SoS could effectively engage or the maximum number of targets that an ISR system could track at one time. In essence this is the scalability of the system defined as the ability of the system to increase the scope of its operations beyond what was initially anticipated. [30]

1.6.3 Space system flexibility

Flexible manufacturing systems are desirable because of the large initial investment required. Similarly, space systems also tend to be very expensive. However, one difference is that they tend to be very inaccessible, such that when a change occurs it is very hard to adapt. Therefore, there is an array of literature on quantifying flexibility for space systems with a greater focus on the system's innate flexibility as opposed to the cost of changing it.

For example, the Galileo spacecraft was launched in 1989. In 1991 as the spacecraft arrived at Jupiter the high-gain antenna, responsible for transmitting most of the data back to earth, failed to deploy, so from 1993 to 1996 NASA to work to redirect the data flow through multiple low gain antennae through extensive reprogramming for data compression and improvements to the ground based signal receiving stations. This element of flexibility allowed the 1.39 billion dollar spacecraft to perform 70% of its mission despite a potentially catastrophic failure. [105]

Shaw [138] defines the flexibility of a satellite by decomposing it into functional modules. Modules are nodes impacting information flow from source to sink in the system. The average cost per node to provide satisfactory service is an aggregate of the cost of designing flexibility into the system. They define two types of adaptability. First is the sensitivity of the performance to changes in technology or requirements. Second, is the ability of the architecture to perform a different mission.

Saleh [135] defines space system flexibility as the ability of a system to respond to changes to its mission in a timely and efficient manner. In contrast, robustness is the system's ability to continue the same mission despite changes to the environment. The distinction is that one measure is focused on the system and the other focuses on its environment. A measure of flexibility is formulated based on the expected net present value of the system as a function of design lifetime. Real options theory is then used to assess the value of maintaining a flexible system.

Real options theory developed by Black & Scholes [27] was developed to assess the value inherent in decision trees. Real options are the ability to make a decision in contrast to monetary options which are investments. It is similar in concept since generally one must invest resources in order to maintain the ability to make certain decisions. The method treats the potential choices in a similar fashion to a portfolio of financial assets. In this way value is assigned dependent on future uncertainty. If uncertainty is high it is worth maintaining a diverse portfolio, but as the future

becomes clearer the cost of keeping an option becomes less than its probable value. This theory is similar to the definitions of flexibility that are based on the number of available states of the system, which is why it is a popular method for assessing flexibility.

However, De Neufville [49] claims that the Black & Scholes equations only work when the product is marketable. If no price can be associated with the product then a reasonable approximation is necessary. Even then it is only applicable if there is a market with established statistics. In the case of space systems this is hard because each system is unique, is intended to last a long time, and is susceptible to major changes during its life.

Nilchiani [115] developed a framework for measuring space system flexibility.

1. Define the aspects of the system that should be flexible. Which parts of the system are most effected by uncertainty.

2. Define the boundary of the system to be studied. Will flexibility be assessed at the component level, or at the system level.

3. Define the time frame in which flexibility will be observed. The nature of flexibility changes with the time scale of the changes to which the system must be flexible to.

4. Identify the sources of uncertainty. The system is being designed for an uncertain and probabilistic future. In order to assign value to the system's flexibility the types of expected changes must be quantified.

5. Define metrics for performance. Flexibility is defined as added value to the system, and the system's value is its ability to perform a certain mission. Therefore, in order to measure flexibility, performance must be quantifiable.

6. Create a baseline and generate alternatives.

7. Assess each alternative's changes to the baseline design.

8. Evaluate alternatives.

9. Explore the trade space.

The basic outline of the methodology resembles the generic systems engineering design methodologies described earlier. The main difference is in how the problem is defined. In this case the focus is on defining only parts of the system that make it more flexible, and the changes to the environment that make flexibility valuable.

For SoS problems, it is possible to define the aspects of the system that should be flexible. However, it is very hard to define the nature of the changes that it will be subject to. If it were easy then it would make sense to define all possible changes as noise variables and simply conduct a robustness analysis. One possibility that follows from the methods described in this section would be to define at least one type of disruption for each category of flexibility. These representative changes could then be used to measure each facet of flexibility. Additionally, while actually using real options theory may not apply here, the idea behind maintaining multiple methods for providing the capability is a concept that seems promising and will be explored later on.

1.6.4 Network flexiblity

A network is a system that is defined by the interrelationships between its components. For systems that are strongly reliant on their ability to pass information or resources between entities, a network is a good way to describe it. Additionally, the SoS's capability can frequently be described by a set of required tasks, the interdependencies of which can also be described by a network. However, the failure of a connection in the network can have repercussions throughout the system, therefore it is valuable to measure the flexibility of the network in terms of its resistance

to such cascading failures. For the maintenance planning problem, the network of maintenance cites and how they share resources is a possible consideration.

One example of a flexible communications network was in the 1950's [11] when AT&T invested enormous amounts of money in upgrading their telephone network to handle the rapidly increasing demand for long distance telecommunications. They found that by allowing the network to evolve, through rapid inclusion of new technologies and dynamic call routing routines they could readily adapt to potentially catastrophic disruptions and increase the overall performance and efficiency of the system.

Cares [35] states the goals of designing networked military forces as stability, adaptability and flexibility. Stability is achieved for networks to which a loss of elements does not greatly impact the performance. Adaptability is the ability for forces to synchronize and reach a steady state of performance. Finally, flexibility is a measure of the range of interoperability options available to the system. The desired result is a military system that would provide stable performance when subject to vigorous adaptations or extreme environments. This statement seems to echo the multi-faceted definition of flexibility stated earlier.

Moses [110] claims that flexibility is an external approach where humans make the changes. Networks are naturally flexible due to the presence of multiple paths, however, this comes at the cost of added complexity as the number of elements increases. In this method, real options are once again used to measure the value of the number of paths through the network. The concept of counting paths through the network is analogous to counting the number of options available to the system.

Continuing on the idea of counting paths, Magee and De Weck [101] define network flexibility as the ratio of paths to nodes in the network. Additionally, flexibility is the ease by which changes can occur and it is defined with respect to network expansion as the ratio of flexibility before and after expansion relative to the ratio of cost before

and after.

$$\text{Flexibility w.r.t expansion} = \frac{\text{links per node after expansion}}{\text{links per node before expansion}} \cdot \frac{\text{original cost}}{\text{expanded cost}} \quad (1)$$

Feitelson and Salomon [62] define network flexibility as the ease by which a network can adjust to changing circumstances and demands, both in terms of infrastructure and operation. They also conclude that flexibility is a multi-faceted metric with the following three categories;

- Node flexibility is the ease by which nodes can be added. Ease is defined in terms of the cost or time associated with the change. Node flexibility is dependent on the ability to physically create a new one such as node size, requirements on node placement, environmental concerns, and interface requirements. Essentially it is like the cost of building a new depot or perhaps the likelihood that a new depot can be placed related to the number of available locations.

- Link flexibility is the cost of connecting two nodes. Just like node flexibility this is mainly related to the requirements placed on the creation of links. However unlike the nodes it is also necessary to consider the traffic on the link and the dependence of a network on links to other networks. For example, air transportation nodes require links to a land transportation network.

- Temporal flexibility is the ability to sequence infrastructure investments and the degree to which the infrastructure requires coordination among users. One part of temporal flexibility is divisibility which is the extent with which choices are locked in. This is essentially, the extent to which paths through the network interconnect or the modularity of the paths.

One criticism of this method is that it does not use quantified measures of the dimensions of flexibility rather qualitative ones. However, the concept of identifying

the cost factors of nodes and links as well as the dependence of the network on other infrastructure is a very good approach. What is nice here is that they define the expected relationships between the dimensions and flexibility. For example, smaller node areas are more flexible as it is easier to move them, more divisible routing leads to higher flexibility and, more requirements means less flexibility. While quantifying the cost of adding a node or an edge is likely a very problem specific quantification, the idea of network divisibility seems generic enough to try. In some sense this is a measure of the number of options available for each required task in the SoS, though if all the options are compatible with one another, a highly divisible network will also tend to have a higher number of possible paths. Therefore, simply counting the viable paths through the network should be a way of measuring divisibility.

Scott et al. [136] propose a method for identifying critical links in a transportation network. This similarity to flexibility is that the ability to identify the weakest link in a network can be useful in quantifying bounds on the system's ability to adapt. The network robustness index evaluates the cost of rerouting around each link. The criticality of the link is dependent on the load which the link is expected to carry. When that link fails, the load must be absorbed by the surrounding paths, therefore, the higher the capacity of the surrounding paths the less critical the particular link will be. The cost of rerouting is assessed as the difference in the cost between the link existing and the link failed. This method is applicable to the maintenance planning problem in that identifying critical paths in the system can be done by evaluating how the demand is shared between the different maintenance sites.

Continuing on a similar idea, Morlok and Chang [109] define capacity flexibility as the ability of a transportation system to accommodate variations or changes in traffic demand while maintaining a satisfactory level of performance. It is measured as the maximum capacity of the system. The assumption is that the greater the capacity the more likely it is to meet the change in demand. Due to the long lead

times required to make changes, it is desirable to design a system that is less likely to need a major overhaul. The method uses an optimizer to determine the maximum load the system can handle. A lower bound on the capacity is evaluated with a fixed network. This corresponds to typical changes in demand and traffic patterns. An upper bound is determined by solving the same problem while allowing slight variations corresponding to more drastic changes such as severe weather or terrorist attacks. If the distribution on demand is known, the maximum capacity can be used to determine the probability that the system will be successful. This concept is similar to the volume flexibility of manufacturing systems, and the idea of quantifying the maximum capacity of the network is likely a good way of measuring volume flexibility for the maintenance planning problem.

In summary, most of these methods focus on the evaluation of paths in the network. Paths are the series of links in the network that information or resources must pass through to get to a desired location. The flexibility of the network is then defined as the ability for resources to continue passing through the network successfully despite changes to the network. The first assumption that is generally made is that disruptions to the network reduce its ability to pass resources. One assumption that can then be made is that the network adapts to restore its performance which can be by adding new elements or rerouting. From this, several concepts of flexibility can be derived, including the divisibility of paths, path criticality and the cost of adding paths through node and edge addition. Alternatively the maximum capacity of the network can be compared to the current demand such that it can inherently sustain a certain range of disruptions without requiring changes. These measures will be considered later on when developing a measure of flexibility for systems of systems.

For the purposes of SoS flexibility, network flexibility possesses one particularly attractive feature that the previous two bodies of knowledge did not, that is, there are a finite number of possible changes to a network. As there are only two categories

of elements in a network, nodes and links, so too all changes to the network can be categorized as either the addition or removal of an element or a change to its characteristics. This is in stark contrast to the infinite types of events that can occur in the real world to a system as large and complex as an SoS. Therefore, characterizing the problem as a network design problem allows all disruptions to be grouped into two abstract categories, which is very desirable.

1.6.5 SoS Flexibility Framework

Building on the methodologies described above, this section will describe the formulation of an SoS framework for flexibility evaluation. It is based on three generic measures of flexibility from the literature. The cost or time delay associated with implementing a beneficial change, the loss of performance due to a detrimental change, and the excess resources in the system that can be used for adapting to changes. The final potential measure of flexibility was a count of the number of available options for performing the capability. For example, the number of missions that an SoS is capable of will make it less costly to add another mission that is somewhat similar. Alternatively, the number of ways that there are to perform a single capability will make it more resilient to disruptions. This kind of redundancy is relevant for the maintenance planning problem as the more available sites there are to perform maintenance the less likely a single disruption will significantly impact performance. However, the number of available options in the system will not be considered an independent measure of flexibility as it can be used to quantify any of the three other measures in some way. Therefore, upgrade cost, resilience to failures, and excess capacity are assumed to be an independent set of flexibility measures.

The literature on flexibility measurement generally requires that the methods of changing the system be defined. That is difficult for an SoS, however if it is described by an abstract network then the number of possible changes is finite. The most basic

change to a network is the addition or removal of nodes or edges. The purpose of an SoS is to provide a capability therefore, the addition of elements will tend to have a positive effect on performance since it is giving the system access to more resources. There will be a cost associated with an added element, therefore this will be categorized as the cost of changing the network. Conversely, removing an element will tend to inhibit the system, therefore it will be considered as a loss of performance due to change. Alternatively, the elements may be rearranged to enable a new capability. This can also be assumed to be a beneficial change. The second category of changes involves the weights on the edges. This will generally effect the magnitude of the flow through the SoS therefore, this will be used to measure the excess capacity of the SoS.

To summarize there are three changes that correspond to three measures of flexibility, and they correspond to the maintenance planning problem in the following ways.

1. Cost of implementing the change

   This applies to the addition or rearrangement of elements in the network. For example, the cost of adding maintenance depots would factor into a measurement of flexibility, or the cost of upgrading components.

2. Loss in performance due to the change

   This applies mostly to the removal of elements in the network. For example, when a depot fails or a route is disrupted the performance will drop while items are prevented from reaching their destinations as efficiently.

3. Excess capacity of the system

   This would apply to changes in the edge weights. For example, when new platforms are added to the system or the operational frequency is increased the load on the network will change and must be accommodated.

From these three categories of changes that apply to a maintenance network, three measures of flexibility can be defined. The purpose of doing so is to define flexibility in the context of the multi-platform maintenance planning problem as well as provide recommendations for how these definitions can apply to other systems of systems.

1.6.5.1 Growth flexibility

Borrowed from manufacturing systems [137], growth flexibility, is a measure of the ease of implementing a change to the system. This is generally done for the purpose of improving the system or adapting to a change in requirements. For example, obsolescence of a component is a change in technology that necessitates an upgrade of the platform. While it is conceivable that such an event will not significantly disrupt the system, it is more likely that an imperfect replacement will be found and the logistics network will have to adapt slightly. In the example that will be described later on, this will result in a change to the location where maintenance is performed. The time it takes for the system to recover from this change is a measure of flexibility. There will be an initial dip in performance as the platforms are removed from service temporarily and new components are purchased, however it can be assumed that a more flexible system will get everything up and running sooner. Another example would be a system that no longer has the capacity to perform due to a change in its operational environment such as a marine task force that has discovered a much more capable enemy than expected. In this case the measure of flexibility will be the time it will take to call for reinforcements. Alternatively, it could be a supply chain that cannot keep up with increasing demand in which case growth flexibility would be the cost of building a new production plant or shipping center.

Graph theory can be used to attempt a generalization of growth flexibility for an SoS network. It can be assumed that the faster a network can reach a steady state the easier it will be to change. Synchronization of a network is the case where all

the elements are interacting properly and on time and the ability of the network to synchronize is how well it reaches a steady state. A synchronized network will reach stability faster and thus more likely to adapt to changes in a timely manner. [124] The second smallest eigenvalue of the Laplacian matrix of the network is also called the algebraic connectivity and is an indicator of the graph's connectivity. It's associated eigenvector is called the Fiedler vector (FV) and is an indicator of how easily the network will synchronize.[120] (see appendix on graph theory for more definitions) Similarly the stability of the network is the the ability to maintain a steady state, which is important as an indicator of how consistently the network is able to provide the desired capability. It can be assumed that it will be easier to change a network that is already stable, therefore, the sum of the absolute values of the eigenvalues of the adjacency matrix of the network is called the graph energy and is a measure of the dynamic instabilities. [16]

The largest real eigenvalue is called the functional cyclicity and its eigenvector is called the Perron Frobenius Eigenvector (PFE). It is commonly used as an indicator of the number of cycles in the graph.[29] Cycles are paths that start and end at the same node, and like paths they can frequently be used to describe the completion of a task chain or the successful transfer of resources in the system. The Coefficient of network effects (CNE) is that eigenvalue normalized by the total number of nodes and is a measure of the networked effects per node which is an indicator of how well distributed the network is.

1.6.5.2 Divisibility

The term is borrowed from the works of Feitelson and Salomon [62], who define flexibility in terms of "the extent to which investments in the network are divisible". This is essentially a measure of the modularity of the system. It is based on the assumption that the more decisions that are available at each point in a process

the more flexible it is. For the maintenance example, consider an architecture with multiple depots that can service the same components. If one depot gets disrupted or has a back log of jobs, broken components can go to the other depots instead and the system continues to function. Another illustrative example would consider a typical morning commute taking the highway. If there is an accident and traffic backs up you risk showing up to work late. However, you decide to bail out at the nearest exit and take side streets the rest of the way. The more possible exits along the way the less likely you will be to get stuck for a significant amount of time in the case of bad traffic. For an engineering design related example, consider the average desktop computer. In the very early days of home computing there were no standard sized components. If a part broke, such as the keyboard, and it was no longer available on the market it was very likely that it would be cheaper to simply replace the entire computer. Now with standardized connectors between each component, there are thousands of options on the market for every part of the computer. If something breaks it is a trivial task to go to the nearest electronics store and buying something with similar functionality.

In the context of systems of systems, divisibility will be defined as the system's resilience to the removal of elements. An SoS's purpose is to provide a capability, which is usually categorized as a series of tasks that allow resources or information to pass through the system. Therefore, it can be assumed that a measure of divisibility would be the number of disruptions required to fully impede the functionality of the SoS. For the maintenance planning problem, this would be the minimum number of depots that the system needs in order to keep the aircraft operational.

From a network perspective it can be assumed that the more paths that the resources can flow through, the more likely it will be that the capability will still be performed even after some number of disruptions. Therefore, an analysis of the number of possible paths will also provide a measure of divisibility. The generic

definition of a path does not distinguish between nodes. However, it is frequently the case in an SoS that the task chain or resource flow must start and stop at a predefined node. For example, information starts at the sensors it is then routed through the command base to the strike assets who use the information to deliver a payload to the target. Therefore, a more specific definition of paths would specify a source and a sink node for the capability. Alternatively, in the maintenance logistics example, the aircraft generally exists at an operational site, however when a part breaks it must go to a depot to be repaired. The capability is only performed once the aircraft returns to the operator. In this case the source and the sink are the same node, which is the definition of a cycle, therefore a measure of cyclicity, defined as the number of complete cycles in the network would make sense. The largest real eigenvalue of the adjacency matrix of a network is called the functional cyclicity and its eigenvector is called the Perron Frobenius Eigenvector (PFE). It is commonly used as an indicator of the number of cycles in the graph.[29] The Coefficient of network effects (CNE) is that eigenvalue normalized by the total number of nodes and is a measure of the networked effects per node which is an indicator of how well distributed the network is. Both of these values may be useful for measuring network divisibility. It should be noted that there can exist systems without defined sources and sinks, such as a communications network where information needs to be able to travel between any two nodes. In these cases a traditional definition of paths should be used.

Unlike cyclicity, there is no indicator of specific paths in the network, therefore the following is an algorithm for counting paths in an SoS network. In general, algorithms for counting paths do not exist since the possibility of getting stuck in a cycle will result in infinite solutions. However, in the case of systems of systems it can be assumed that getting stuck in an infinite loop is not going to actually happen, since the agents involved will realize and set themselves back on the correct path. Therefore a modified depth first search algorithm is proposed for counting paths from

all source nodes to all sink nodes in a network. A depth first search decomposes the network connections into a tree and searches down each branch of the tree. [60] The modification that is added is that when a node is repeated the branch is terminated so as not to consider cycles.

1. Start at the source
2. End a branch when the sink is reached or when a node is revisited
3. Explore all possible branches
4. Count the number of branches that end at the sink
5. Repeat for all combinations of sources and sinks

Having counted the paths it is now possible to examine their criticality to the system. One possibility is by assessing the cost of rerouting should a path be disrupted. It would follow then that the measure of criticality would be related to the number of paths that pass through a certain edge. Additionally, if the capacity of the edge and the expected load on it is known then these can be taken into account as well. It will be assumed that when a path is disrupted that the load that was initially intended to pass along it will be redistributed across all other available paths. The criticality of the edge can be defined as the ratio of the load on the path to the number of alternative paths that do not use that edge. For edge i with load L_i, there is a set of paths P that accomplish the same capability. The number of paths that use edge i is p_i, therefore the number of paths in P that don't use edge i can be called p_i^o and the criticality is $\frac{L_i}{p_i^o}$.

Alternatively, a modified definition of connectivity can be used here to count the minimum number of elements that must be removed before no possible path remains. This path connectivity problem can be reformulated as a min-cut problem between

a source node and a sink node, where all edges have value 1. This way the value of the minimum disconnecting cut will be the total number of edges removed.

1.6.5.3 Volume flexibility

Also borrowed from manufacturing systems [137], volume flexibility refers to the ability of the system to handle a wider range of demands than it is expected to experience. Consider the marine task force again. In the case where they encounter a larger enemy force than expected, a system with higher volume flexibility would already possess the necessary resources to handle the larger situation and wouldn't need to call for reinforcement. Alternatively, in the case of the aircraft maintenance system, when the frequency of maintenance actions increases such as when the operators transition from peace time operations to war time, volume flexibility describes the extent to which the system is already capable of handling the change. This can be easily quantified as the difference between the maximum capacity of the system and the current expected demand. One way to measure this would be to find the level of demand that causes the system to cease functioning properly. Alternatively, if it is assumed that the capability can be quantified by the flow of resources through the system, then the max-flow in the network (appendix A) may be a good generic measure, where the edges of the network represent the capacity to pass resources between entities. Additionally, if a distribution is known for the demand on the system then the probability that the demand exceeds the maximum capacity can be calculated using the cumulative distribution function of the demand.

$$F_v = P(\text{demand} < \text{maximum capacity}) \tag{2}$$

1.6.5.4 Section Summary

In conclusion, it can be hypothesized that including a measure of flexibility in an evaluation of an SoS will result in designs that perform better in the presence of

disruptions and changes. To do this, three categories of flexibility are defined that primarily apply to the multi-platform maintenance planning problem, but are also generalizable to other SoS problems. The first category is growth flexibility which is defined as the ease of implementing a change in the system. The second category is divisibility which is defined as the system's modularity which is closely related to the system's resilience to the loss of elements. Finally, volume flexibility is defined as the range of capabilities that the system can perform. For the maintenance system, growth flexibility is the cost of adding depots or the cost of upgrading a component, divisibility is the number of depots that the system can lose before failing, and volume flexibility is the maximum number of platforms or the maximum operational frequency that the system can reliably handle. Experiments will be conducted in chapter 4 to show how these three flexibility measures relate to the cost and performance of the system when disruptions are considered and to determine whether these measures are redundant or not.

1.7 Modeling and simulation

Given a set of objectives for the design process the next step is to define a method for evaluating those objectives for each alternative system architecture. In general, actually implementing each design and testing it is too expensive and time consuming therefore, systems engineers tend to rely on mathematical models. As George E.P. Box is famously known for saying "all models are wrong, some models are useful." [32] In order to evaluate designs, implement the methodology, and conduct experiments a modeling method must be chosen for the multi-platform maintenance problem. The field of modeling and simulation is very broad, therefore the purpose of this section is to review some of the literature in this area in order to justify the choice of methods that were used for this thesis.

First, to review some definitions, the DoD defines a model as "a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process". Modeling is defined as "the application of a standard, rigorous, structured methodology to create and validate a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process", and simulation is "a method for implementing a model over time". [52]. Ideally when developing a model, it would be attractive to match the details as closely as possible. However, including too many details may make determining the cause of a particular behavior difficult, due to confounding between factors. In reality, the act of modeling is the act of walking a fine line between the amount of detail and the ability to draw meaningful conclusions.

Since the problem chosen for this thesis is notional, it would be wise to err on the side of fewer details in order to make the results more generalizable. As mentioned earlier, the primary assumption that differs between this work and previous studies is the consideration of disruptions to the system. Where previous work could consider a static aggregative model of service costs, this study intends to look at the effects of one-time events on system performance. Gustafsson [80] categorizes simulations into four categories distinguished by either stochastic or deterministic behavior and static or dynamic time scales. The discontinuity in performance caused by disruptions implies that a dynamic model must be used which requires the use of a simulation. Additionally, the exact nature of the disruptions is unknown therefore they must be modeled as random events. In general, the presence of stochasticity introduces variance into the response, which requires that each case must be repeated in order to obtain a statistically significant sampling of the model. This indirectly increases the computational cost of the method. However, deterministic simulations tend not to capture as much of the behavior. If the deterministic model can be defined in such a way as to capture sufficient behavior then it is the preferred method. For the

time being though, it must be assumed that the disruptions are random events, so in order to sufficiently capture that behavior, a stochastic/dynamic simulation must be chosen.

Balestrini [18] identifies several simulation options for the purpose of modeling complex military systems. Since, complex military systems tend to exhibit many SoS properties, the following list should apply well to this problem.

- Network

  Network models focus on the connections between entities in the system. Most of the methods for evaluating network properties involve assessing the entire network at once thereby requiring the assumption at any given point in time the network is static. Stochasticity can be introduced fairly easily by assigning random distributions to the weights of the edges, or by randomly removing and adding elements to the network. Network models tend to be good descriptions of systems of systems, but their static nature makes it difficult to use them for evaluating objectives. However, in some cases graph theory has be shown to efficiently evaluate certain network properties that sufficiently capture the behavior of the system.

- Dynamical systems

  Dynamical system simulations use ordinary differential equations of the system to model its behavior [33]. As such they tend to be best suited to physical system where the dynamics are well understood and where the number of differential equations is relatively small. Similar in both name and concept, systems dynamics models are also essentially a system of differential equations. However, the key difference is the inclusion of feedback and its effect on the aggregate behavior of the system. It was originally designed by the engineering controls community for application to business strategy.[69, 70] For systems with many

moving parts and complex behaviors defining the exact differential equations is likely to be very difficult, therefore this method is unlikely to be useful here.

- Discrete event

  In a discrete event simulation (DES) the system state is updated at discrete time intervals corresponding to the occurrence of specific events.[161, 75, 76] It is particularly well suited to model supply chains and manufacturing processes, where it can be assumed that no two events truly happen at the same time, and that nothing particularly interesting happens in between events. [81, 150, 158] A maintenance system shares many characteristics of a supply chain therefore this is likely to be a good method for this study. Additionally, disruptions are easily described as discrete events making DES a good option for modeling disruptive environments.

- Markov chain

  Based on the Markov principle that the current state of the system is only dependent on the previous state, Markov chain models represent the transition logic between discrete states in the system. As a generalized version of a discrete event simulation, Markov chains require the enumeration of all the possible states of the system and their transition probabilities. As such they are ill suited to modeling systems with a large number of state variables, such as having many discrete entities moving around.

  Similar to Markov chains in that it represents state transitions of the system, Petri nets [127] consider the states of individual entities and resources in the system which are represented by tokens moving on the net. Stochastic petri nets [160, 17] have been shown to be good for modeling reliability and resource management, and are much better suited to large problems than Markov models.

- Agent based

A cellular automata model is constructed of a grid of cells where each cell can have one of a finite number of states. The states change according to a set of rules that check the states of the neighboring cells. For example, in Conway's game of life, each cell is either "alive" or "dead" and looks only at the eight neighboring cells when determining when to change state. There are three rules corresponding to overcrowding, under-population, and reproduction.[44] The goal of the simulation is to observe emergent behaviors in the system, therefore it is best suited for problems with large populations of interacting stationary entities. [102].

Agent based simulations are similar to cellular automata but they add a layer of complexity by decoupling the entities from their environment. In this way the model considers mobile agents and stationary patches that each manage their own state transitions by following a given set of rules. Having the assumptions implemented from the bottom up means that they are capable of representing systems in near complexity to the system itself. [19]. As such they are well suited to modeling unpredictable systems that are strongly dependent on the independent decision making capabilities of individual agents such as combat systems. Given that maintenance depots can be considered stationary and are not expected to make complex decisions, agent based modeling may be a bit excessive for this thesis.

There are two other models that were specifically used to simulate systems of systems.

- Aggregation models

  Iacobucci [86, 85]proposes a framework that is designed to handle large system of systems alternatives spaces. In order to handle the entire design space, the generality of the model must be increased which means that it ignores many of

the details that help define a unique system of systems. The Rapid Architecture Alternative Modeling (RAAM) method provides the capability to evaluate large architectural alternative spaces in a reasonable amount of time, and to a level of fidelity appropriate for conceptual phase decision making. The backbone of the RAAM computational model is based on aggregation functions that combine subtask performance metrics to obtain higher level task metrics, and continuing on until one overall score remains. While it is fairly typical to decompose a SoS capability into a series of tasks and subtasks, the problem is that the methodology provides no criteria by which to create the aggregation functions.

- Information theory models

  ARCNET was developed by Domerant [53, 54] as a simplified engagement model to be used within the ARCHITECT methodology. It was developed for the purpose of modeling SEAD architectures and take into account the effects of collaboration between systems on the overall performance. The basic principle upon which the model is based is derived from Perry's method [125] which uses information theory to quantify the knowledge in a C2 system. The method starts by quantifying the mission uncertainty with a probability distribution, then using information entropy to measure the "average amount of information" in the distribution. This can then be related to the overall probability of success of the mission, which is a useful metric of effectiveness for a military architecture. Since collaboration between systems generally involves sharing information it can be assumed that collaboration will affect the average amount of information in the system which will affect the overall performance. The result is a function relating the mission performance to the architecture. While ARCNET is a full engagement model where information theory is used to quantify the probability of finding and successfully engaging an enemy unit, the method can be expanded to apply to a larger scope.

To summarize, the linear programming models for traditional LoRA are not sufficient, because they fail to model the effects of discrete changes to the system such as obsolescence or depot failure. Therefore, a dynamic simulation is preferred. Lacking a well defined model for how the changes will happen requires that the simulation be stochastic. Of the methods listed by Balestrini, a network model could be used to evaluate designs but might not be detailed enough to capture meaningful measures of system value. Therefore, discrete event modeling was chosen due to its suitability for supply chain modeling, a problem that is fairly similar to maintenance logistics. However, systems of systems are generally dependent on a large number of independent decision making entities, meaning that an agent based model will be the appropriate simulation method the majority of the time.

1.8 Discrete Event Simulations

DEVS, Discrete EVent Simulation is a formal method for modeling systems where the state only changes as a result of an "event". The system is then modeled by only considering the events and not the static time in between. [161]

The simulation requires four elements:

1. A discrete set of state variables

   For this problem each component has a number of state variables associated with it including its condition(functional, broken, or obsolete), location, and remaining lifespan. If it is a component then its location is tied to the platform that it belongs to which is tracked as a state variable instead.

2. Simulation time

   Every time an event occurs the simulation time counter is updated to match the time associated with that event. By tracking the total time a maximum simulation time can be set. For this problem the simulation tracks time in

days and simulates 10 years worth of operations. It is assumed that this time span would allow for each component to go obsolete at least once, given that computer components are generally only on the market for 5 years [126].

3. A synchronizer to maintain the list of events

   The synchronizer it the primary driving force in the simulation. It iterates through the list of events, one at a time, in chronological order, and runs the appropriate functions associated with each event. After it is done it updates the time counter.

4. Events representing state changes

   Each event represents a different set of state changes governed by its own logic. For this problem, the basic flow of logic can be summed up as follows. Platforms exist mainly at the operators until at some point a part breaks. The platform with the broken part is then shipped to a depot to receive a working replacement at which point it is sent to the warehouse to await future operations. This is because in the meantime the operator has requested that a working platform be sent from the warehouse to replace the one with the broken part. Once the broken part has been removed from the operator it is determined whether it can be repaired or not. If it is repairable it is sent off to be repaired otherwise it is discarded. After being repaired it is held at the depot until another platform arrives that needs a replacement of the same type. Figure 16 summarizes that behavior.

   That described the typical behavior of the system, events that trigger due to state changes internal to the simulation. However disruptions are also modeled as discrete events. For example obsolescence causes all the components of one type to change state. Other types of disruptive events would an increase in operational frequency, depot failure or the procurement of additional platforms.

The technical difference between disruptive events and regular events is that the disruptive events are not necessarily triggered by state changes within the system rather they are determined by criteria external to the simulation. As such, external events must be defined by the analyst either specifically or randomly during simulation initialization.

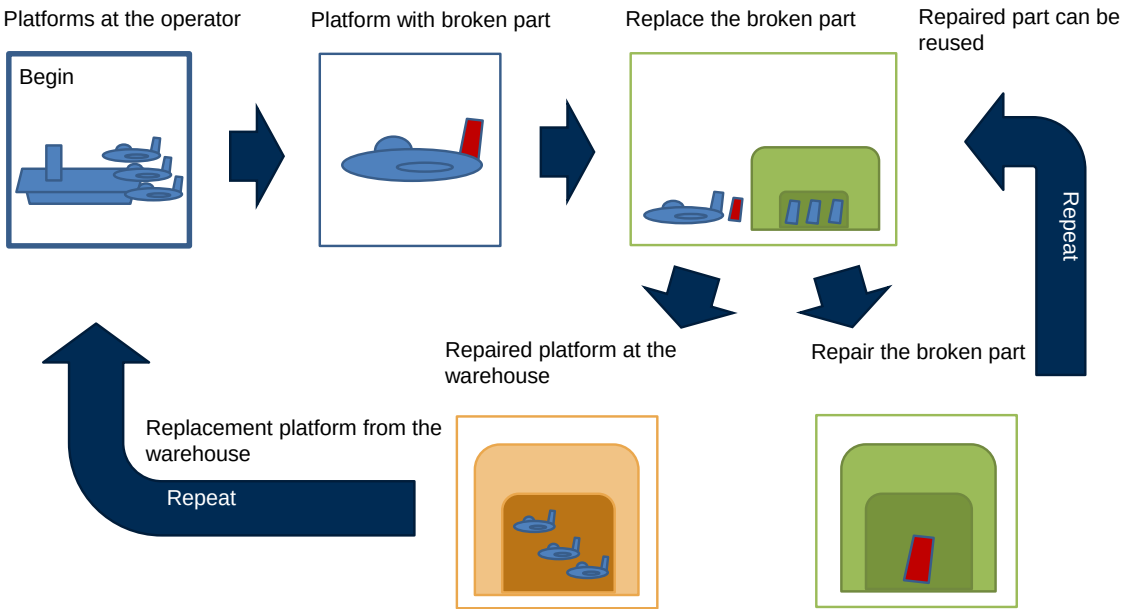


Figure 16: Overall logic flow of FLoRA DES

At this point it is sufficient to state that a discrete event simulation can be formulated for assessing the system's performance, cost and flexibility. Chapter two will go into much greater detail on the formulation of the Fleet-wide Level of Repair Analysis Discrete Event Simulation (FLoRA DES).

1.9 Heuristic evaluation

For small scale problems using a discrete event simulation or an agent based simulation may be feasible. However, typical runtimes for these types of models range from a couple second to several minutes per case. For higher fidelity versions the computational cost can be even greater. In general for a system with the complexity

level of an SoS the number of moving parts and decisions that must be considered at each time step in the simulation requires an enormous amount of computational power. Given the expected size of the design space, even a simulation that takes a fraction of a second to run will result in a design space exploration that takes months to finish. While a few months is frequently acceptable, increasing the scope of the problem even slightly can result in an analysis that could take lifetimes. For example, some military capabilities require on the order of one hundred different tasks. If the decision making process is reduced to simply two choices for each task the resulting design space has 2^{100} (about 10^{30}) alternatives. Given that even the best simulations on the best computers can only do several thousand runs per second, even a very optimistic estimate of the total run time (between 10^{15} and 10^{20} years)would be greater than the age of the universe (13 billion years). The same logic would apply to a maintenance planning problem with 100 different component, each of which has two possible maintenance options. In general, this issue of excessive simulation costs is a very real issue for large scale complex systems. Therefore, if the goal is to implement an architecture as quickly as possible analysis methods that take years to complete are not preferable, rather there must be an alternative to computationally expensive dynamic simulations.

Cares [36], in discussing complex adaptive military systems, suggests that "when assessing complex systems, it is not possible to know the odds, regardless of the amount of information," he recommends that "it is better to understand a system's dynamics". This recommendation leads one to believe that no matter how well one models the system's behavior it will always deviate in some un predictable way. The best that can be done is to develop indicators of favorable emergent behaviors and design for those. This would indicate that lower fidelity methods would be acceptable here.

As mentioned earlier, traditional LoRA prescribes the use of heuristics to qualitatively down select the number of alternatives before using more quantitative computational methods. In this way the number of architectures that get fully evaluated is down selected in order to minimize the computational cost. To define heuristics, "heuristics are strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines".[123] In essence solving a problem with a heuristic is the process of using less than optimal solutions, experience, and time saving approximations to make a decision when using more accurate methods is not efficient enough. It is a trade off between optimality, accuracy and precision for efficiency and speed. As Koen states in his definition of the engineering method,"The engineering method is the use of heuristics to cause the best change in a poorly understood situation within the available resources."[94] This is doubly true of SoS problems where they are never fully understood and the available resources are never truly enough. Therefore, if suitable heuristics can be formulated than they could be used to perform the initial design space down selection before higher fidelity simulations get used. While rules of thumb and qualitative guidelines frequently make good heuristics, the goal of this thesis is to develop a quantitative and repeatable method.

The field of computational problem solving has a number of ways of using heuristics to solve problems that are too big for brute force. Some example problems include, the 8-puzzle, and the traveling salesman problem.

1.9.1 8-puzzle

Most children are familiar with this simple puzzle game. (figure 17)

The goal is to arrange the tiles in order from 1 to 8 with the blank spot in the corner by moving one tile at a time. Since the average number of moves to solve the puzzle is 22 and the average number of possible moves at any step is about 3, a

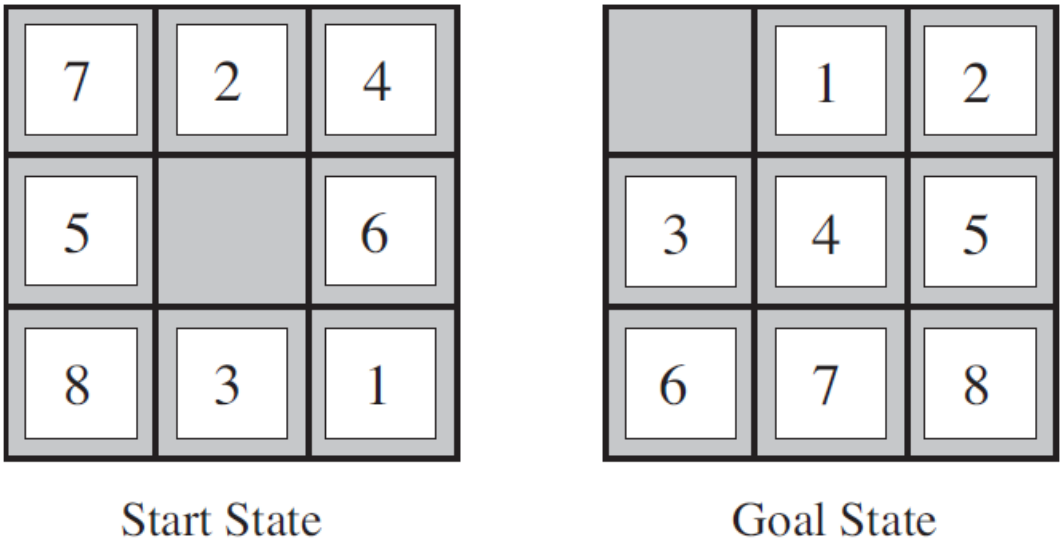


Figure 17: Initial state and end state of the classic 8-puzzle

search through every possible path would have $3^{22} \tilde{} 10^{10}$ different possibilities. The problem to be solved is what is the minimum number of moves required to solve the puzzle from a given state. This is similar to SoS design problems in that evaluating each solution is computationally difficult and there are too many possible solutions to evaluate them all in a reasonable amount of time. Therefore heuristics are used to get a solution that is close enough. Two commonly used heuristics are used for this type of problem by solving a simplified version of the problem with relaxed constraints.

1. Removing the constraint for the tiles to stay on the board allows one to solve the puzzle by moving each tile exactly once. The solution is then equal to the number of tiles out of place.

2. Alternatively by removing the constraint that only one tile be moved at a time, allows one to solve the puzzle by moving each tile equal to the number of spaces it is distant from its desired location. Therefore the solution is equal to the total displacement of all tiles.

The heuristics are used to reduce the number of solutions fully evaluated by discarding solutions that are heuristically poor. While far from accurate in terms of the

actual number of moves required to solve the puzzle, both of these heuristics provide a conservative estimate on the solution, and algorithms implemented with them have been shown to perform several orders of magnitude better than the brute force approach in terms of the total number of alternatives evaluated and the computation time. [133]

1.9.2 Traveling salesman problem (TSP)

Another classic computational problem, that is solved using heuristics is the traveling salesman problem. TSP models the shortest path a salesman could take to visit every city on a map. To demonstrate how large the solution space is, a TSP with 20 cities has 20! possible solutions which is about 10^{16}. The accepted heuristics for solving this problem revolve around the method for searching the solution space. For example, one could use a greedy heuristic and assume that starting at any city the next city to visit is the next closest one. This is a simplified version of the problem and tends to perform well up until the last few steps which tend to be less than optimal. Alternatively a better heuristic would be that at each city, choose the next city based on a TSP solution of the remaining cities. This heuristic is determined by solving a sub-problem, the TSP of fewer cities, and tends to work better than the first option. [133][65]

For any problem where the solution space is too large to evaluate directly, heuristics can be developed to make the search viable. This can be done in one of two ways: relaxing the problem or solving a sub-problem. Either way as long as the heuristic is easier to compute than the true solution, and is reasonably related to the actual value, the resulting search will be much more efficient. Therefore, the next question is what types of heuristics are suitable to SoS evaluation and more specifically to the multi-platform maintenance planning problem.

To summarize, the first possible source of heuristics is by asking the subject matter

experts (SMEs) that would've been making the down selection decisions themselves. Their qualitative input could be used to inform a quantitative set of rules to use in estimating the performance of each solution. The problem with this is that SMEs capable of providing that kind of information for an SoS are few and far between, making this approach unreliable at best and certainly not repeatable.

Moving on, the field of computational problem solving suggested two possible methods for developing heuristics. The first method is solving a sub-problem or only solving part of the problem. This would indicate one of two options: either only evaluating some of the metrics, or only running a partial simulation. If the metrics are sufficiently unique then only evaluating a partial set of them will result in a poor design space analysis, therefore running partial simulations would be a more feasible method here. The basic formulation of such a measure would be to run the simulation until a point where it can be sufficiently determined that the performance will likely be strictly worse than another option, similar to the way split times are used in downhill ski racing as an indicator of where on the leader board the current skier is on pace to finish. In this case it should be possible to discount most of the solutions by arguing that they are unlikely to succeed in the long run without fully evaluating them. However, at best this is only likely to reduce the computational cost by one order of magnitude, and does not guarantee the multiple orders of magnitude improvement in evaluation time that is promised by the use of heuristics. The second method, simplifying the problem, can be accomplished by evaluating a simplified version of the SoS. Although such a model would tend to be overly abstract, it might give a reasonable estimation. It was mentioned earlier that a static network model could describe an SoS but would not provide high fidelity estimates. However, such a model would rely on graph theory to calculate network properties which may relate to the system's performance and would be several orders of magnitude less computationally costly than dynamic simulations.

To summarize, the requirement on a heuristic measure can be stated as follows:

- Partial description of the problem
- More computationally efficient than the other available methods

The possible available methods for generating heuristics are:

- Ask subject matter experts
- Evaluating a simplified model
- Partial simulation
- Partial evaluation of metrics

Of the four options, a simplified model would be ideal if it could be shown that such a model sufficiently represented the problem. One such abstraction of an SoS that has shown some merit in the literature is the network of connections between the actors in the system. This makes a fair amount of sense given that an SoS is defined as a set of independent entities that must work together to achieve a goal. A network is a mathematical description of those interactions that can be evaluated using graph theory, which is more computationally efficient than a dynamic simulation. Therefore, it can be hypothesized that network properties are heuristic measures of SoS value and can be used in place of computationally expensive dynamic simulations.

Chapter 3 will describe a network based framework for measuring SoS flexibility. This will be used along side the simulation described in chapter 2 to conduct the experiments in chapter 4 comparing simulated measures of performance to network properties. If a relationship can be found, then the network properties will be used to conduct a design space exploration and hypothesis 2 will be considered substantiated.

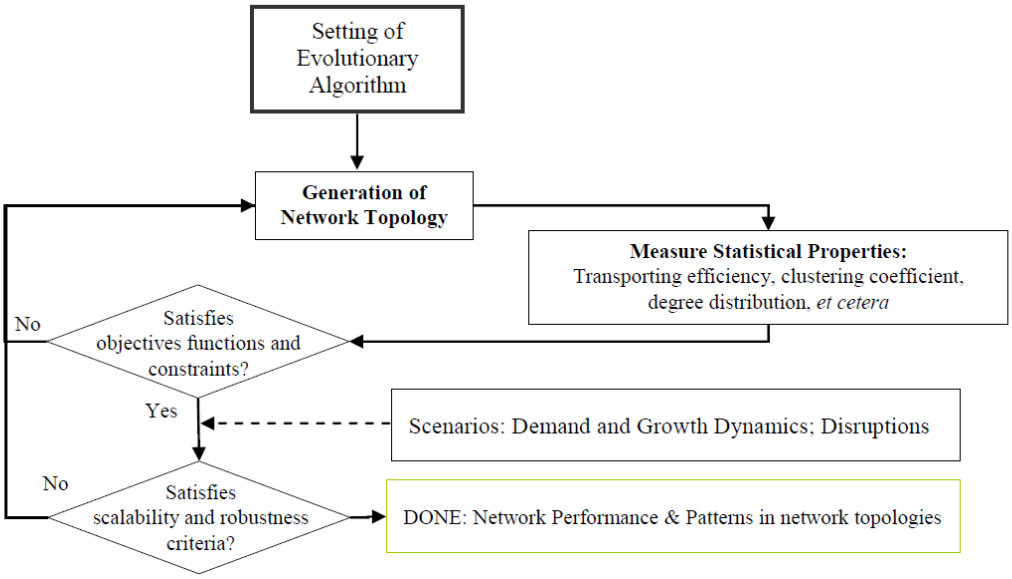


Figure 18: Design-oriented network analysis framework

1.10 Network modeling

If networks are to be used to describe systems of systems, then a guidelines for doing so must be defined, therefore the following will be a brief survey of the literature on SoS network modeling. The purpose of doing this will be to gain insight for how a network model of the multi-platform maintenance system can be formulated.

1.10.1 A Network Theory-based Approach for Modeling a System-of-Systems

Han and Delaurentis [82], applied graph theory to the study of aviation transportation networks. The design problem was to design a network that is cost efficient, robust and scalable. The approach was to introduce network connectivity analysis into an optimization framework. An evolutionary algorithm was used to explore the space of feasible network topologies while each network was evaluated for performance as well as subjected to a set of disruptive scenarios to determine whether the network satisfied the robustness and scalability criteria. (Figure 18)

As a case study, a measure of transportation efficiency was developed that is the

normalized average weighted shortest path of the network, which in layman's terms is a measure of how the demand on the network is distributed over the possible paths, similar to the measure of divisibility described earlier. Additionally, centrality measures were used to identify critical airports in the North American air transportation system. The study concludes by calling for more work to be done in identifying a set of evaluation metrics for SoS networks. The value of this study is that it suggests that there are, in fact, network properties that relate to the flexibility of the system.

1.10.2 A Modeling Process to Understand Complex Architectures

Balestrini [18] used graphs as a description of the activities that link together to provide a capability. He states that "network models are one of the most extreme forms of models, in that they simplify and abstract to the maximum level possible, yet retain the essence of the system being studied, and can therefore help in understanding that system's behavior and characteristics." Cycles in the graph represent whether the capability is achievable or not. Graph theory is then used to calculate a set of network metrics that are used to compare two alternative systems of systems and determine with high confidence which one will perform better. Since the focus of the evaluation is on the cycles in the graph, the measures used are the functional cyclicity, CNE, PFE, node cyclicity, and the Fiedler vector, which are indicators of the number of cycles in the graph, and the stability of the network. In addition, the most important nodes were identified for the purpose of determining which systems require the most attention when developing a modeling and simulation environment. The nodes were ranked by, PFE, FV, indegree, outdegree, and clustering coefficient. The rankings were tested on random boolean networks, and it was found that the inverse PFE and absolute FV performed the best. (see Appendix A for graph theory overview) In summary, this work compiled a list of generic graph theory measures that may relate to the system's performance.

1.10.3 ARC-VM: An Architecture Real Options Complexity-Based Valuation Methodology for Military Systems-of-Systems Acquisitions

Domercant [53] developed a measure for SoS architecture complexity using network properties. Based on the assumption that the complexity is a result of systems collaborating and sharing resources, an analysis of the underlying network was logical. The measurement is broken into four sub-measures

1. Functional distribution complexity is the distribution of functionality across the unique systems. A functional integration exponent is used to quantify the difficulty involved in integrating the systems. One approach is to quantify this by the connectivity. [148]

2. Functional process complexity measures the distribution of systems across the functions. This is quantified by the number of possible paths through a program control graph of the process, via a measure called cyclomatic complexity. [108]

3. Resource state complexity measures the interactions based on the number of resource exchanges. One method for quantifying this effect is to determine how the resource sharing network will react to local disturbances. A number of measures can be used including PFE, CNE, CPL, clustering, graph energy and algebraic connectivity.

4. Resource processing complexity is a measure of the interactions based on the magnitude of the resource exchanges. It is quantified in the same way as the resource state complexity, except using an adjacency matrix weighted by pairwise interoperability instead of a matrix of need-lines.

   To summarize, this work used a similar set of graph theory measures as the previous work, but also gives some guidelines as to how the network models should be formulated.

1.10.4 The Information Age Combat Model

Cares [37] uses a graph theory approach to analyzing network operations. He proposes that any military force can be decomposed into four categories. Sensors receive signals and relay the information to the deciders. The deciders assess the information and determine the present and future arrangements of the other nodes. The influencers act on the information received from the deciders. Finally, the targets do not generate or use information but none the less are still important. The resulting information flow network can be used to study how different network metrics relate to the behavior of the military system of systems. This decomposition of networked entities is primarily relevant to military systems, however the concept categories of entities should be applicable to the maintenance planning problem.

1.10.5 Department of Defense Architecture Framework

Architectures are a convenient and documented way of describing a system. An SoS architecture is defined in ANSI/IEEE 1471-2000 [9] to be "The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." The Department of Defense Architecture Framework (DODAF) [116] was created to aid in design and decision-making for large and complex systems and systems of systems in a capability-based context, by providing a repeatable and documented method for describing the systems. The framework is divided into categories of architectural views: all-view, operational view, systems view, services view, standards view, data and information view, project view, capability view. The views are intended to describe everything from a graphic overview of the functionality of the system, the resource sharing dependencies, the list of tasks and who must do them, all the way down to the technical details on the communications systems. Many of the architectural views are network descriptions, such as some of the operational views and system views.

These views can be used to inform a network model of the SoS. In general though, the problem with using architectures for a design space exploration, is that they tend to be better for communicating how a SoS works, than for generating alternatives or performing automated evaluations.

1.10.6 Dynamic network analysis

There is literature on dynamic network analysis [38, 40, 39] that suggests the use of a network with probabilistic edges for describing complex, dynamic organizations. It is best suited for use with social networks that are in flux such as terrorist organizations. An SoS can be considered a dynamic organization in the sense that it exists in a changing environment and must evolve and adapt, therefore this analogy makes a certain amount of sense. In this method, the organization is described by a meta-matrix which is a set of linked networks. Changes in one network are assumed to cascade into the other networks. Therefore the focus of the analysis is on metrics derived from multiple cells in the meta-matrix. Each network in the matrix is a combination of the relationships between agents, information, tasks, and organizations. The relevant sources of change in the system are identified and the corresponding cells in the meta-matrix are chosen to evaluate. Once the interactions between each network are defined then they can be evaluated using traditional network evaluation methods, such as graph theory.

The meta-matrix methodology is based on a previous method by Krackhardt and Carley [98] for modeling the structure of an organization. The organization is decomposed into three types of elements: agents, tasks and resources. Agents are the actors in the organization, the people or groups that use the system. Tasks are actions that the agents take in order to generate value from the organization. They could be steps in a process intended to provide a capability, or they could be independent actions taken by the agents to improve their position in the organization. Finally,

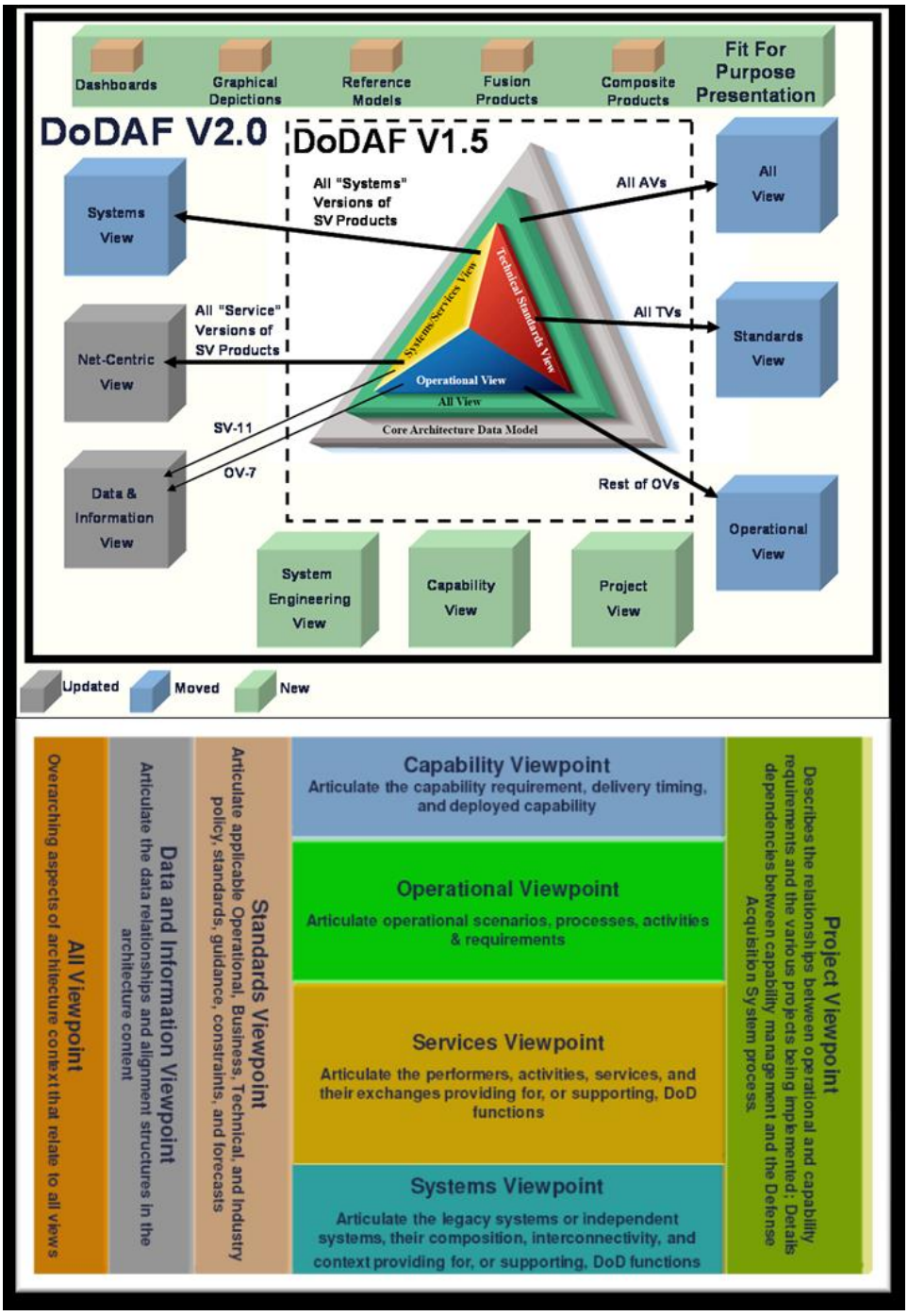


Figure 19: DoDAF overview [116]

Table 1. Networks of agents, knowledge, tasks and organizations

	Agents	Knowledge	Tasks	Organizations
Agents	**Interaction Network** *Who knows who* Structure	**Knowledge Network** *Who knows what* Culture	**Assignment Network** *Who is assigned to what* Jobs	**Employment Network** *Who works where* Demography
Knowledge		**Information Network** *What informs what* Data	**Requirements Network** *What is needed to do what* Needs	**Competency Network** *What knowledge is where* Culture
Tasks			**Precedence Network** *What needs to be done before what* Operations	**Industrial Network** *What tasks are done where* Niche
Organizations				**Inter-organizational Network** *Which organizations work with which* Alliances

Figure 20: Design-oriented network analysis framework [38]

resources are objects of value in the organization. They can be physical resources like aircraft or spare parts, information, or even skills that are possessed by the agents.

The three sets of elements are then used to generate a multi-layered network description of the organization. The precedence matrix (P) is a network relating tasks to each other. Generally it describes the order of the task chain such that if task i is followed by task j then $P_{ij} = 1$. The commitment matrix (C) relates tasks to resources such that if resource j is required or designated for task i then $C_{ij} = 1$. The Assignment matrix (A) relates agents to task in a similar way to the C matrix. The network matrix (N) is called as such due to relationships between people as being the most common use of the term network. It describes the interrelationships between agents. This can represent friendship, work groups, resource sharing and many more relationships. Finally the skill matrix (S) relates agents to resources, describing who possesses which skills.

The interesting aspect of this methodology is that the basic matrices can be combined using matrix multiplication in order to generate even more detailed descriptions. Even though three of these five matrices are not square, any matrix can be multiplied by its transpose because the inner dimensions will match. For example AA' will be a square matrix which describes which agents share a task in common. Similarly PP' will describe the dependencies of tasks on one another. Since P is a square matrix APA' is a valid matrix and combines the information in both matrices and describes which agents are dependent on which other agents to finish their tasks.

1.10.7 Section summary

First, multiple layered networks will likely provide a more detailed description of the SoS than a single network model. As a general rule, the number of different possible networks will be all the possible combinations of the different types of entities that are being modeled. $(\frac{n^2-n}{2} + n)$. For example, the PCANS method [98] considered

three types of entity, resources, tasks and agents, therefore there are a maximum of six different matrices. However, Krackhardt considers the resource to resource relationship to be irrelevant, therefore the PCANS method uses five matrices. Using this style of method, each matrix can be used to describe a number of different relationships, therefore it is up to the analyst's discretion to use as few or as many network descriptions as they like. For the design of systems where architecture definition is a required product, it would make sense to use something like DoDAF to inform which networks to generate.

With regards to the minimum number of different entities that should be modeled, it can be assumed that agents and tasks must be differentiated. Since a capability is defined by a set of tasks it is logical to require that network model. Additionally, an argument can be made that agents of some sort must be modeled in order to determine the performance of specific tasks. As for resources, it is sometimes the case that they change hands so rapidly that the time scale on changes in the resource based networks are sufficiently small that they don't provide meaningful insights on the problem. This is the case with maintenance systems where the resources are the actual platforms and spare components. These change location so frequently compared to the rate at which depots change or components go obsolete that the resource to agent relationships are transient in comparison. However, in the PCANS method, resources correspond to skills or knowledge possessed by agents, which tend to be more static, therefore a network model of resources makes more sense in such a scenario. To summarize, tasks, agents and resources can be considered the basic building blocks of an SoS network model. However, it is always possible to model different types of elements separately, making the possibilities endless.

Finally, when measuring the flexibility of the SoS it is worth being careful as to which network measures are used and how the edge weights are defined. First, probabilistic edge weights are good for measuring dynamic parts of the system. For

example, when counting paths for the purpose of measuring divisibility, a probabilistic approach would consider the likelihood that a path exists given a set of possible changes. Second, when measuring volume flexibility, it is best to define edge weights by the time or capacity that it will take to move along that edge, such as the mean time to repair a component and ship it to the next site. In this way the measurement of resource flow through the network makes more sense. Third, a task chain is in essence a model of the steady state of the system, therefore the spectral properties relating to stability and synchronization in the network can logically be applied here. It should be noted though that spectral properties are only valid on a square graph, as the eigenvector problem only has a unique solution for square matrices. Only networks relating like type entities (ex: agents to agents) will result in square matrices.

1.11 Design space exploration

Another alternative to evaluating all possible designs with a simulation would be to be more selective of the designs that do get evaluated. This would fall into the category of strategic down selection or design space exploration techniques. Many of the methods described in this section can be used along side heuristics for even greater effect.

1.11.0.1 Full factorial

The most basic of design space exploration methods is the brute force method of just evaluating all the possible solutions. The problem with this is that for design spaces with continuous variables there are technically an infinite number of solution. Similarly, for problems with extremely large design spaces such as the maintenance planning problem, the amount of time to evaluate the entire design space could take many years, even with a very fast model. However, for fully discrete design problems of a manageable size, this is clearly the best method.

Iacobucci[86], attempted to develop a method for performing a full factorial analysis on SoS design spaces called the Rapid Architectural Alternative Modeling (RAAM). The basic idea was to decompose the SoS's main capability into a hierarchical tree of sub tasks, down to the leaf task which are performed by single systems. In theory, the performance values for these leaf tasks are known and independent of the architecture of the SoS since they are single system problems. These values are then aggregated by a series of functions until only a single high level SoS metric of effectiveness remains. The aggregation functions are supposed to account for the interactions between the systems in the effort of performing the capability. The result is a blazing fast evaluation of each architecture that can be used very early in the design process to evaluate billions and even trillions of architectures in short order. Technically speaking, a tree is a type of graph which has no cycles, therefore this method has the added bonus of possibly making good use of the network theory discussed previously. However, the biggest issue with RAAM, is that a method for developing the correct aggregation functions was never defined. When the task turned out to be too difficult, RAAM was abandoned as a viable method.

1.11.0.2 summary

1.11.0.3 Monte-carlo simulation

The next logical step would be to evaluate a fraction of the design space and then down select from that set, only keeping the solutions that are not strictly worse than any others. If the evaluated set is chosen randomly, the actual best solution is assumed to be in the neighborhood of the best evaluated solutions, therefore a refined search can be conducted on these smaller spaces. However, this assumption falls apart when the design space is discrete and discontinuous. When it is discrete, there is not necessarily a relationship between a solution and its so-called "neighbors" and the discontinuities ensure that there is no way to predict where the actual best solution will be based on a random sampling. Alternatively, a Monte-Carlo simulation

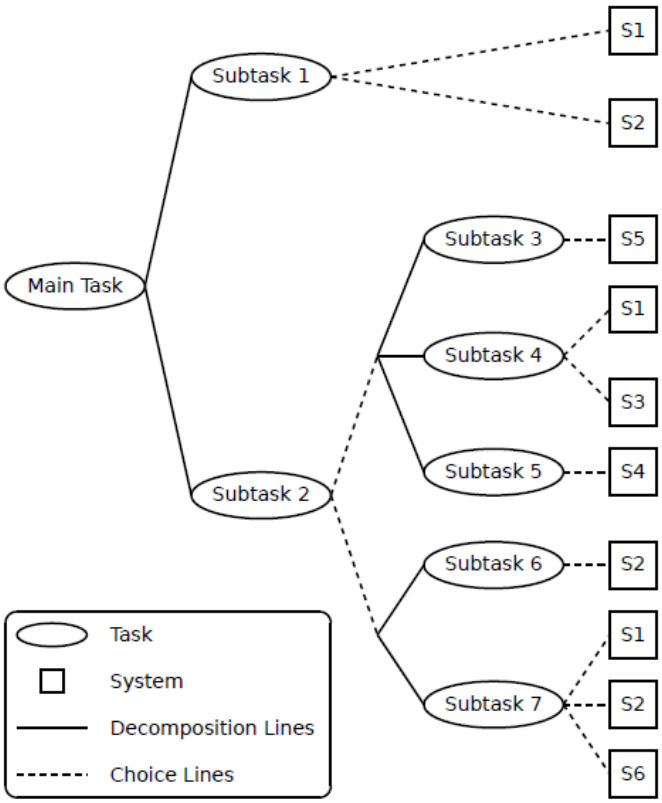


Figure 21: Example of task decomposition for RAAM evaluation

assigns probability distributions to the design factors, and the resulting distribution is a reasonable approximation of the probable performance of the final product. The problem with this approach is that for very large design spaces there is no good way to ensure that all areas of the design space are sufficiently sampled, and since the results are dependent on the input distributions, for SoS problems where those distributions are frequently unknown, the results make no sense. [51] However, random sampling can work at times and for the time being will be considered as a baseline method of design space exploration.

1.11.0.4 Design of Experiments(DoE)

Now consider that one intelligently chooses which solutions to evaluate. A DoE attempts to ensure that the design space is sufficiently sampled from the perspective of the design variables. The results are then statistically fit to a surrogate model, so that the performance of the solutions that were not evaluated can be estimated. This method was applied to a combat SoS problem by Ender and Biltgen [26]. A simulation was developed to evaluate the overall performance of the SoS and the design space was sampled by varying the design parameters of the individual systems in the architecture. Surrogate models were made using response surface equations and neural networks so that trade-off studies could be done. The major difference between their problem formulation and the multi-platform maintenance planning problem is that they assumed that the architecture was fixed and proceeded to investigate the effects of changes to the individual systems in the SoS. However, when the architecture is the focus of the design problem, since it is assumed that most of the component systems already exist and cannot be changed, the resulting design space is very different and the use of a DoE and surrogates makes less sense. Once again the discrete nature of the design space makes this difficult, as a DoE functions by sampling the design variables within continuous ranges. If the design variables are discrete then

the only way to sample the design space is to evaluate every discrete alternative and no time is saved. Additionally, while there are methods that can generate surrogate models for discrete inputs and discontinuous outputs, the idea of emergent behavior implies that the performance of the SoS does not necessarily follow from the sum of systems that went into it. [111] The assumption of emergent behavior implies that no strong statistical relationship should exist between the SoS architecture and its performance, which negates the primary purpose of surrogates. Therefore, if the goal is to generate heuristics, the relationship that must be explored is the relationship between different sets of output metrics, such as the relationship between network properties and flexibility measures.

1.11.0.5 Algorithmic design space exploration

The previous three methods all choose to evaluate solutions regardless of whether they are good or not, and no decisions are made until after all the cases have been run. However, when it is possible to use heuristics to assist the search, a more selective approach can be taken using an algorithm to systematically search for better solutions. These heuristic approaches are most commonly used for optimization in an attempt to systematically move through the design space iteratively improving the solution as they search. The result is that the algorithm stops when it can no longer improve the solution any further, which in theory will be the best possible solution. In doing so, these algorithms promise to evaluate a much smaller fraction of the possible designs than other methods. In general there are two types of approaches, those that evaluate one solution at a time and those that work on a set of solutions. [65] Ultimately, the goal here is to find a method that can handle discrete design variables, potentially discontinuous outputs, multiple objectives and is well suited for very large design spaces.

The first family of single solution algorithms, evaluate the whole solution each

time. The most common heuristic for this family of algorithms is to always move in the direction of a better solution. The direction of best improvement is usually determined by taking the gradient of the objective function, and the algorithm stops when there are no more "up hill" directions to pursue. There are two problems with this approach. First, the computation of the gradient tends to be very computationally costly, and the gradient may not exist in the case of discontinuous objective functions. Second, in the case of a function with many local optima, or if there are multiple objectives, the algorithm's inability to go "down hill" means that there is no guarantee that the result is the global optimum. Even if the algorithm is simplified to not use the gradient, there is no way to escape the second issue with a hill climbing algorithm, aside from running it multiple times starting from different initial points. [157]

The other family of single solution algorithms attempts to build a solution one design decision at a time. These algorithms use a heuristic to determine what the next best step is, such as a greedy heuristic, or by recursively solving the remaining sub-problem as described in section 1.9. There are once again two main issues. First, it requires a logical starting point otherwise the result will depend on that initial decision. For example, the branch and bound method [99] starts from the optimum of the continuous space to systematically partition the space in order to find the optimum within the discrete settings of the continuous design variables. Alternatively, the traveling salesman must visit all the cities at some point, therefore the TSP algorithm can start at any city. The second issue is that the algorithm must be able to evaluate a partial solution at every step. However, in the case of most SoS problems, the evaluation method is usually a complicated M&S environment where a partial set of inputs makes no sense. [65] In summary, no single solution algorithms are likely suitable for the multi-platform maintenance planning problem, or SoS problems in general.

The second family of heuristic algorithms uses a set of solutions, or a population,

to converge on the best possible design. As a result of seeing more of the design space at once, the algorithm is less likely to get stuck in local optima. The differences between these types of algorithms mainly depend on the way they vary the solutions and the way they share information between members of the population. One subset of these algorithms can be called swarming algorithms, because they model the search through the design space as a swarm searching for the best solution. For example, the most basic particle swarm algorithm[92] allows the members of the population to determine their velocity based on the velocities, locations and best known solutions of each other member in the swarm. Alternatively, the search can be modeled after bees looking for pollen [128], where some bees are responsible for finding good search locations, and others are responsible for exploiting those locations, or fireflies looking for mates[163], where the perceived luminosity of each firefly is a function of distance and the value of the objective function, and each firefly is attracted to others that are the most luminous. The problem is that these methods require a measure of distance between solutions, which in the SoS case is not always possible due to the discrete nature of the design space.

Villeneuve[159] applied an ant colony algorithm[55] to a discrete system design problem. In this approach, the design space is modeled as a graph where each node represents a design decision and the edges are the other feasible design decisions remaining. Therefore a path from one end of the graph to the other represents a complete design. The search is modeled after a colony of ants searching for food, where each ant leaves pheromone along the path that it took to the food. The amount of pheromone is proportional to the quality of the solution found, and future ants will tend to follow a similar path. The problem with this approach is that it is very hard to introduce any knowledge of the problem into the search, since such heuristics would be used to guide the ants along the path, and therefore require the evaluation of partial solutions to compute.

The other set of population based algorithms model the search after the theory of evolution [15]. These evolutionary algorithms (EA) assume that the members of the population compete against one another to advance to the next generation, in the same way as the theory of survival of the fittest. The differences between EAs come from the different ways there are to represent a solution, vary the solutions, evaluate the fitness of solutions, and select the composition of the next generation. For example, the classic genetic algorithm[84] represents the solution as a string of bits, and creates new solutions by combining the bit-strings of two parent solutions with high fitness. Alternatively, the algorithm can be used to evolve a computer program[96] by representing the solution as a series of functions applied to the inputs, and variations are done by adding, removing or combining sets of functions. As such, EAs are very flexible due to many different options available when constructing one.

1.11.0.6 summary

Figure 22 summarizes the different options available for design space exploration. Full factorial would work if given infinite time, and a DoE to make surrogates would work if the design space were less discrete. Therefore, as a baseline, random sampling is the only non-algorithmic method available. As for the available algorithms, traditional approaches also struggle with the discrete and discontinuous nature, as do most swarming algorithms, which leaves evolutionary algorithms as the best remaining option. They are extremely flexible, and easily adaptable to most problems if the right configuration is used. To make an analogy, EAs are like the tofu of the optimization world in that they can be used for any problem as long as the right seasoning is used. This leads to a conjecture that evolutionary optimization is an appropriate method for handling large discrete and discontinuous design spaces such as systems of systems.

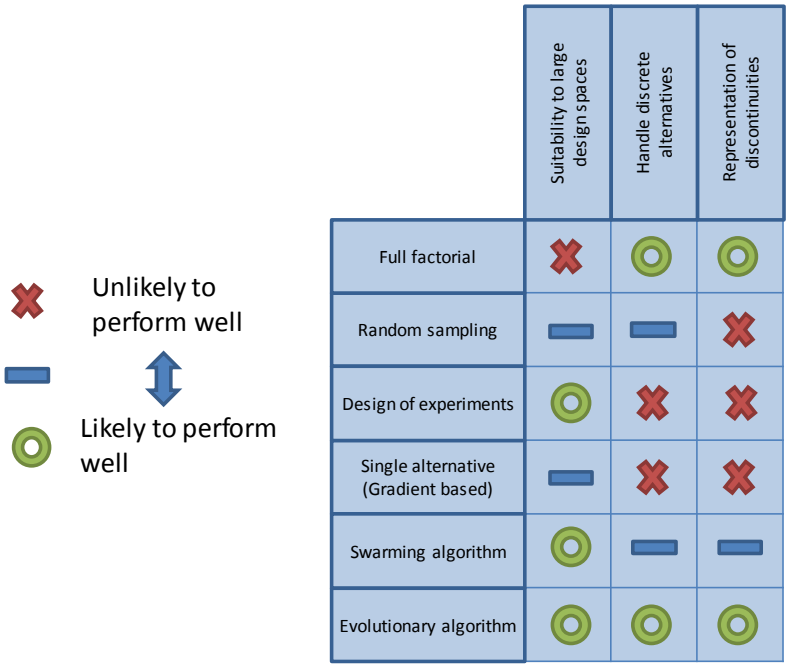


Figure 22: Summary of design space exploration methods

1.12 Evolutionary algorithms

Given that EAs are extremely versatile, some thought must go into how an EA should be configured. Ultimately, the choices will depend on the nature of the problem, but some generalizations can be made for most SoS design problems. Each EA is essentially comprised of the following five functions.

- Solution representation
- Population initialization
- Evaluation
- Selection
- Variation.

1.12.0.7 Solution representation

For the purpose of representing the design space in a way that the EA can understand, there are two options that make sense for this problem.

- Binary string

  Each design variable is encoded in zeros and ones, then the bit-strings are concatenated to represent a single solution. This type of representation is particularly efficient for problems with a lot of boolean variables such as, do task A or not, or system A and B are linked or not. Binary strings are also well suited for design variables with discrete settings, such as the maintenance site where a particular component will go to get repaired.

- Real value vectors

  Alternatively, each design variable can be represented by its actual value instead of converting it to a binary string. This is a more logical representation for continuous variables such as a component's failure rate or a depot's labor rates.

Fogel and Ghozeil [66] proved that for representations that are bijections (representations that can be translated from one to another), there is no advantage in choosing one type over the other, therefore it makes sense to choose something intuitive. For this problem the network topology is easily represented by its adjacency matrix. This is essentially a real valued matrix, which the EA can handle just as easily as a vector. For the system specific parameters, some are boolean and some are not therefore, it makes sense to use a real valued vector.

1.12.0.8 Population initialization

Unless guidelines are used to initialize the algorithm with more favorable solutions, the only remaining option is to choose a random population. However, there is one guideline that makes a lot of sense in the context of systems engineering design problems. The baseline solution, if it exists, should always be in the starting population. This way an argument can be made to the customer that the resulting best solution is both an evolution of the baseline and exceeds its performance. [65]

1.12.0.9 Evaluation

The evaluation step of the EA is where the performance metrics are translated into a measure of solution fitness. For single objective problems, the model evaluation can be used directly as the fitness function. However, for this problem the model will return multiple metrics which must be aggregated in some way for fitness evaluation. The available methods can be separated into three categories as follows.[68]

1. Weighted sum

   Each metric is given an importance factor and the fitness is the weighted sum of the metrics. Additionally constraints can be handled via threshold functions on the weights such that if a metric does not meet the constraint then its weight is set to zero. A hard threshold sets the entire fitness value to zero if the constraint

is violated.

2. Single criterion

   Unlike in the final decision making process where all metrics must be considered at the same time, an EA will run through many generations making minor decisions at each point. Therefore, it has the freedom to consider one criterion at a time. Among the possibilities are that the EA cycles through the metrics one per generation, uses a randomly selected metric at each step, or even a randomly chosen linear combination of the metrics. Alternatively, if the metrics have discrete settings then a lexigraphic approach can be used, where the metrics are ranked and the solutions are then sorted in a similar way to alphabetization.

3. Pareto based

   The Pareto front of a design space describes all the solutions that are not strictly worse than any other solution. The remaining solutions are considered "dominated". The simplest form of this would be to simply sort the solutions by non-dominated ranking. The non-dominated solutions are removed and given rank 1, then the next set of non-dominated solutions are removed and given rank 2, and so on and so forth.[74] The problem with this is that for problems with more than two metrics, the number of rank 1 solutions can be very large therefore, there are a number of ways of increasing the granularity of the ranking. One way is to determine rank by the number of solutions dominating it within the population. [67] Another way is to use a hybrid method which ranks the solutions within each Pareto rank using one of the above two methods. The issue with increased granularity is that it tends to favor certain regions of the Pareto front. However, early in the design process, when preferences are still in flux, it makes the most sense to sample the Pareto front evenly.

To summarize, without subject matter experts to determine the weighting scenario or at the very least rank the metrics, only Pareto based methods are viable. Additionally, since the goal is to down select the design space and keep a small set of optimal solutions for further analysis, a method that considers multiple solutions equally is desirable. Therefore, for the maintenance planning problem and most likely SoS design problems in general, Pareto based methods are ideal.

1.12.0.10 Selection

Once the fitness has been evaluated, the members of the current generation that will advance to the next generation or at least be allowed to vary must be determined. There are two types of selection operators, deterministic and stochastic, and the only functional difference between them is the amount of selective pressure applied to the population. This relates to the rate at which the population will lose diversity, with stochastic methods tending toward slower convergence.[14] It can be assumed that slower convergence favors problems with large design spaces and many local optima as it allows more time for the algorithm to find the best solutions. In general though any of the the following methods will work well.[15]

1. Proportional

   The solutions advance to the variation step proportional to their fitness compared to the rest of the population. If done deterministically, the proportions are used to determine the exact make up of the advancing population. If done stochastically, the proportions are used as a probability distribution of the advancing population. [84]

2. Tournament

   Pairs or groups of solutions are compared, and the better solution is considered the winner and advances. Alternatively, many tournaments can be conducted with replacement, and the advancing solutions are the ones with the

most "wins". This tends to be a stochastic method because, the competing pairs are usually determined randomly.

3. Rank-based

   Instead of using fitness to determine the probability of reproduction, the solutions are ranked ordinarily and are chosen to reproduce as such.

4. Elitism

   While not a true selection operator, it is a possible addition to any selection operator. In this case the N best solutions are automatically advanced to the next generation and are exempt from the rest of the selection process and variation as well. This is a way of remembering the current best solution. It was shown that elitism improves the performance of multi-objective genetic algorithms by preventing the loss of good solutions. [166, 132] However, it does not always work well with Pareto based evaluations where there are usually many elite solutions.

1.12.0.11 Variation

Finally, in order for the algorithm to explore the design space it must have an operator for varying the population. There are two options for this as well.

- Mutation

  Each design variable is given a random chance of mutating. If it mutates, then a randomly distributed value is added to that variable. If binary strings are the chosen representation, then this is usually done on a bit to bit basis and instead of adding a value, it is simply flipped.

- Crossover

  First, two parent solutions are selected, then some number of their variables are swapped to make two offspring solutions. This is modeled after the way genetic

material is combined in the process of mating. The offspring get half their genes from the mother and half from the father. The theory behind crossover's effectiveness is called the building block hypothesis [84] . It states that the algorithm identifies segments of the genome, or building blocks, that tend towards good solutions, and recombine them until the population converges.

The random nature of both variation operators is one way to ensure that the algorithm is likely to find the global optimum or at least get very close, therefore either is likely to work. However, consider the idea of emergent behavior in a SoS. It essentially states that the whole does not equal the sum of the parts, which seems to be in contradiction to the building block hypothesis that says that the genetic makeup of the solution, or genotype, determines the optimality. The alternative is to look at the phenotype, or the manifestation of the genes, to determine optimality, which seems to be more consistent with the problem definition.

1.12.0.12 Non-dominated Sorting Genetic Algorithm II

To summarize, for this design problem an EA that will return a good sampling of the Pareto front is preferable. Other than that there were no strong preferences for what other options should be chosen. Additionally, given that the values that will be used for the design case study in chapter 5 will be mostly notional, a perfectly tuned algorithm is not necessary. Therefore, in the interest of not reinventing the wheel, a well documented EA that has been shown to perform admirably will be used. For that, the Non-dominated Sorting Genetic Algorithm II is a multi-objective genetic algorithm that was developed to reduce the computational complexity of other non-dominated sorting algorithms, include elitism, and eliminate the reliance on sharing parameters to maintain Pareto front diversity. It was shown to find a "much better spread of solutions and better convergence near the true Pareto-optimal front compared to other multi-objective EAs. [50] Additionally NGPM (NSGA-II

Program in Matlab) [145] is an NSGAII software package compatible with Matlab that interfaces very nicely with the discrete event simulation described in chapter 4.

1.13 Flexibility based SoS design methodology

To summarize the chapter, a methodology for designing flexible systems of systems can be proposed. It is built onto the skeleton of a methodology described in section 1.5 by filling in the gaps with knowledge of the following three hypotheses.

1. Including a measure of flexibility in an evaluation of an SoS will result in designs that perform better in the presence of disruptions and changes.

2. Network properties are heuristic measures of SoS value and can be used in place of computationally expensive dynamic simulations.

3. Evolutionary optimization is an appropriate method for handling large discrete and discontinuous design spaces such as systems of systems.

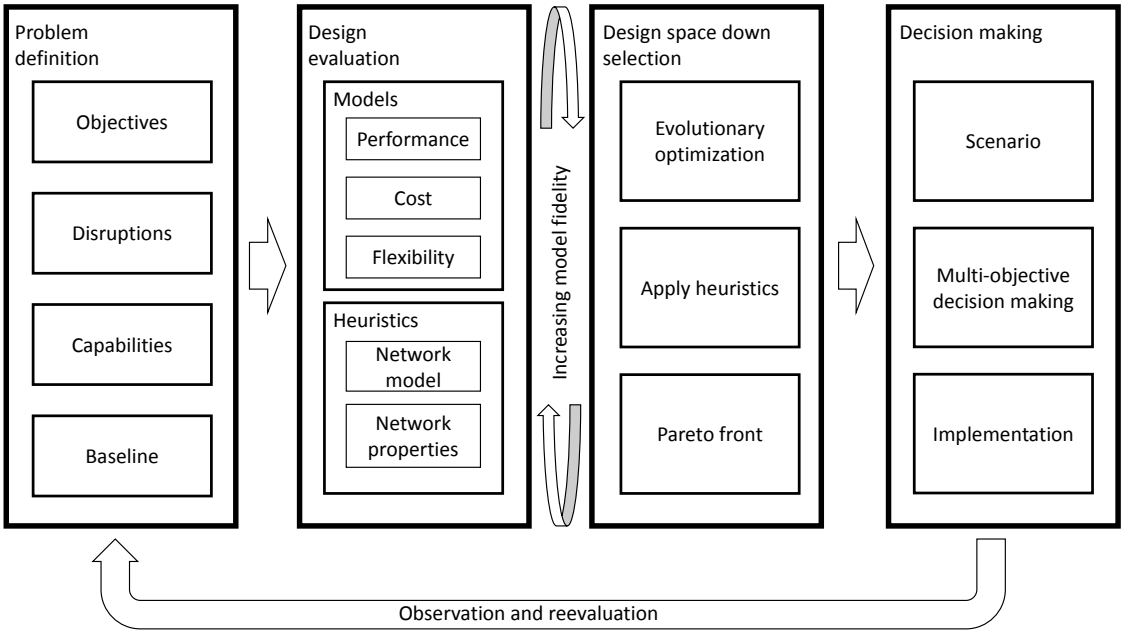


Figure 23: Methodology for designing flexible systems of systems

The methodology is made up of four steps that are repeated at intervals for the lifetime of the system. The four steps are as follows.

1. System definition

   The boundaries of the system must be defined, including all the component systems and how they are intended to behave, and what kinds of capabilities the system should have. Then, the expected disruptive conditions must be considered. While it is likely impossible to fully list all the possible disruptions, a representative list will be used to evaluate the system's flexibility. Finally, the baseline of the system must be defined. These are the attributes of the system that must remain fixed, everything else will be allowed to vary and can be used to generate alternative designs.

   To relate back to the maintenance planning problem, the system is defined by a network of operators, depots and warehouses that all work together to keep the fleet of aircraft functional. For this study, the problem has been scoped down to only consider the maintenance of the on board mission computers of the aircraft. As for disruptions, it is expected that component obsolescence, increases in force structure and operational frequency, and the loss of depot could have serious impacts on the performance of the systems. Finally, while the primary objectives are to minimize O&S costs and maximize availability, flexibility is considered as well for which three different measures were defined.

2. Design evaluation

   This step describes the modeling process where computational simulations of the system are developed in order to assess the quality of potential designs with respect to the objectives. In addition to traditional models of cost and performance, a model for flexibility is developed here that is used in evaluating the system's response to disruptive events. Finally, since it is expected that

these simulations will be very computationally inefficient, the second half of this step involves the formulation of heuristic measures. It was hypothesized that network properties would be suitable heuristics for the flexibility. Chapter 2 is a documentation of the discrete event simulation that was developed for the multi-platform maintenance planning problem, and chapter 3 is a documentation of the corresponding network model.

3. Design space down selection

   In this step an evolutionary algorithm is used to efficiently find the set of non-dominated designs. This Pareto front is then set aside for further investigation. If the reduced design space is still too large then the methodology returns to the previous step for a higher fidelity model and repeats the optimization process until the design space is sufficiently small.

4. Decision making

   Decisions can be made and the chosen system architecture is implemented. This is done by using knowledge of the current scenario in order to tradeoff between the different objectives and pick the design that is best suited to the given customer's priorities. Lacking a real customer for the multi-platform maintenance system or the resources to implement it, the analysis done in this document will not include this step.

5. Observation and reevaluation

   Finally, since the design of an SoS is never truly complete, once the system is implemented it must be continually observed. Hopefully, by considering flexibility, the system will be able to handle most disruptions and changes. However, at some point it is expected that the system will have evolved past the point where it no longer truly resembles the original design. At this point, the design

methodology should be revisited and the system reevaluated for the next time a change needs to be made.

CHAPTER II

FLEET WIDE LEVEL OF REPAIR ANALYSIS DISCRETE EVENT SIMULATION (FLORA DES)

2.1 Model Overview

The Fleet-wide Level of Repair Analysis Discrete Event Simulation (FLoRA DES) tracks the state of items in a maintenance logistics system. The model assumes two levels of indenture.

1. Platform

   The simulation calls them platforms in order to be generic, and can represent any part of the end item, but in this case it is the complete mission computer box for a single aircraft. Due to the one-to-one relationship between mission computers and aircraft it is unnecessary to consider higher levels of indenture. However, if multiple mission computers can go in an aircraft then a third level of indenture must be modeled.

2. Components

   The lowest level of component considered by the model are the modules of the mission computer. Failures happen on this level as does obsolescence and repairs. In some LoRA models the mid level component can themselves fail however, this model assumes that all components are considered, therefore only the lowest level of component can fail.

Information describing the components and platforms is tracked in the THING global variable. Each entry in the THING vector possesses the following variables.

Table 2: Contents of the THING global variable data structure

Name	Item ID number is generally the index in the THING vector
condition	"functional", "broken" or "obsolete"
repair (component)	The location ID number of the depot where it will be repaired
replace (component)	The location ID number of the depot where it will be replaced
operator (platform)	The location ID number of its operator
mtbf (component)	Its mean time between failures
prodCost	item unit cost
life (component)	The total amount of operational time measured in days
inopsince	The simulation time when the item most recently began operating
super (supercomponent)	The platform that a component belongs to. If the item is a platform or a lone component then super = name
components (platform)	Vector of component ID numbers
type	the type of item (example: 'C1', 'P2')
weight	Item net weight
obsolete (component)	Boolean variable, 1 if the item is obsolete
failure (component)	The amount of operational time that this component will actually last for

location	Location ID number where the item can currently be located

To add an item to the THING vector the GenThing function is used. The item is added to the end of the THING vector, life is set to zero, and failure is set to a value randomly sampled from the exponential distribution with mean MTBF. Location must be set manually.

```
GenThing(name, repair, replace, operator,...
     MTBF, production cost, type, components, weight)
THING(end).location = loc;
```

Information describing each component type is stored in the PARTS global variable. Each entry in the PARTS vector possesses the following variables.

Table 4: Contents of the PARTS global variable data structure

type	component type name (example: 'C1', 'DH8x10')
unitCost	procurement cost of the component
mtbf	failure rate
weight	component net weight
repair	the level at which this component should be repaired
replace	the level at which this component should be replaced

Information describing each platform type is stored in the PLATFORMS global variable. Each entry in the PLATFORMS vector possesses the following variables.

Table 5: Contents of the PLATFORMS global variable data structure

type	Platform type name (example: 'P1', 'F18C')
components	Vector of PARTS index numbers of each component that is required for this platform

The model considers three echelons of maintenance in addition to the storage warehouse.

1. Operator
2. Intermediate level of maintenance
3. Depot
4. Warehouse

Information describing the state of the locations is tracked in the "PLACE global variable. Each entry in the PLACE vector possesses the following variables.

Table 6: Contents of the PLACE global variable data structure

Name	Location ID number is generally the index in the PLACE vector
mttr	The mean time to perform a maintenance action
queue	Vector of item ID numbers representing items that are awaiting maintenance. For the warehouse a vector of location ID numbers representing operators that are awaiting replacement platforms
labor	Location labor rate

type	0 - warehouse, 1 - operator, 2 - intermediate, 3 - depot
spares	Vector of component ID numbers representing spare parts
coordinates	x & y coordinates
nspares	The number of spares that the location ideally possesses
partsserviced	The index in PARTS of each component that can be serviced at this location
fhPerYr (operator)	The operational frequency of the operator measured in flight hours per year
force	Vector of platform ID numbers representing operating platforms at an operator or spares at the warehouse
functional	Boolean variable, 0 if the depot has failed

To add a location to the PLACE vector the GenServer function is used. The location is added to the end of the PLACE vector with no spares and no jobs waiting in the queue.

```
GenServer(name, labor, MTTR, type, coordinates, nspares, partsService, fhPerYr)
```

The simulation models meaningful behavior using events. Each event represents a different set of state changes governed by its own logic. For this problem, the basic flow of logic can be summed up as follows. Platforms exist mainly at the operators until at some point a part breaks. The platform with the broken part is then shipped

to a depot to receive a working replacement at which point it is sent to the warehouse to await future operations. This is because in the meantime the operator has requested that a working platform be sent from the warehouse to replace the one with the broken part. Once the broken part has been removed from the operator it is determined whether it can be repaired or not. If it is repairable it is sent off to be repaired otherwise it is discarded. After being repaired it is held at the depot until another platform arrives that needs a replacement of the same type. Figure 24 summarizes that behavior.

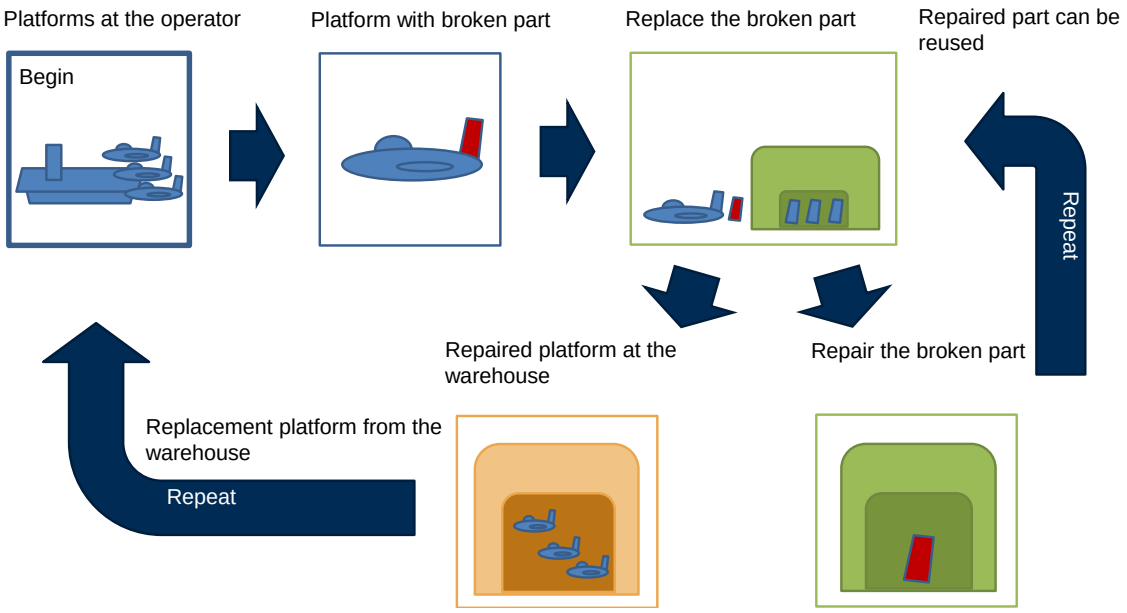


Figure 24: Overall logic flow of FLoRA DES

Information describing each event is tracked in the EVENTS global variable. Each entry in the EVENTS vector possesses the following variables.

- type

  The name of the event so the simulation can call the correct event resolution function. The different types of events are as follows and are described in more detail in future sections.

  - repair
  - failure
  - discard
  - ship
  - replace
  - blockbuy
  - obsolete
  - upgrade
  - fieldmod
  - failDepot
  - operational_shift

- who

  The entity that the event is referencing. Can be a location ID number, item ID number or an item type string

- time

  The simulation time at which the event will occur

- cost

  The cost associated with this particular event

- amount

  Some events require that an amount be specified

- location

  Some events require that a location ID number be specified

- availability

  The platform availability at the time that the event occurs

To add an event to the EVENTS vector the GenEvent function is used. The function maintains the chronological order of events by adding the new event at the correct index. That index is returned by the function for reference by the simulation. The amount and location must be set manually while availability is set when the event resolves. It should be noted that some events also set cost upon resolution instead of generation.

```
N = GenEvent(type, who, time, cost);
EVENTS(N).location = x;
EVENTS(N).amount = y;
```

Event resolution is controlled by the synchronizer which is a function that iterates through the list of events, one at a time and runs the appropriate functions associated with each event. After it is done it updates the time counter which is stored in the TIME global variable and assesses the availability to be saved for post processing.
In order to set up the initial configuration of the simulation the following input data structures must be defined.

- platforms = {type, components}
- components = {type, maintenance plan, MTBF, weight, unit cost}
- operators = {location ID, force structure, fhPerYr, labor rate, nspares, MTTR, parts serviced, location}

  Force structure is a vector of length equal to the number of platform types where each entry represents the number of platforms of the corresponding type that the operator should get.

- depots = {location ID, parts serviced, MTTR, labor rate, nspares, location}
- intermediates = {location ID, parts serviced, MTTR, labor rate, nspares, location}
- warehouses = {location ID, force structure, location}
- external events = who, type, time, amount, cost

  External events are events that are not triggered by any other event, rather they are predetermined before the simulation to occur at a specified time.

The set of initialization functions is then run to set up the scenario specified by the input file. It performs the following operations.

1. Parse the input file to generate data structures that the simulation can understand.
2. Determine each service location's parts serviced list based on each component types maintenance plans.
3. Initialize all locations
4. Initialize platforms according to each operators force structure and schedule their respective failure events.
5. Initialize spare platforms at the warehouses
6. Initialize spare components at each service location
7. Add external events to the event list.
8. Set TIME = 0 and start the synchronizer at event #2, since the first event is a summary of initialization.

At this point the simulation can be run with the predefined stopping conditions. There are four options for stopping condition as follows.

1. Maximum simulation time, which is the natural stopping condition representing the life span of the system measured in number of days.

2. Minimum availability, which represents the time of system failure used for measuring flexibility. For example, if the system is considered failed when availability drops below %65 then 0.65 would be used here. If no minimum value is desired then a negative value should be used in order to prevent this stopping condition from triggering. This is because zero availability can occur while negative availability cannot.

3. Maximum number of events, which can be used to specify a condition for identifying infinite loops in the simulation. If the simulation is working properly this value should be set to infinity, since guessing the total number of expected events can be difficult.

4. Errors in the code will cause the simulation to exit unnaturally. At the time of writing this document there were some errors that occurred infrequently enough that they were too hard to catch and fix. For example, there were some errors that occurred approximately once in every 1000 runs of a scenario. When the sample sizes are only expected to be in the 100-200 case range, it is practical to skip the failed cases and move on. Therefore, when running the simulation in repetition mode, the error status output should be checked and all faulty cases should be discarded.

2.2 Description of individual functions

2.2.1 Repair

Table 8: Repair function summary

Variable name	Value
type	repair
who	Component ID number
time	TIME + event duration
cost	Event duration * Labor rate
amount	N/A

```
t = randexp(MTTR);
GenEvent('repair', 4 , TIME + t, t*laborRate );
```

This is a fairly simple function since most of the complicated logic is handled before the component ever arrives at this state. The cost and duration of the repair job is determined based on the location's MTTR and labor rate at the time that the event is scheduled.

$$\text{repair cost} = \text{labor rate} * \text{event duration} \tag{3}$$

$$\text{Event duration} = randexp(MTTR) \tag{4}$$

To repair a component the model assumes that repairing the component essentially resets its life span. Given the original failure time which is the current age of the component, the model randomly assigns a new failure time using the exponential distribution with mean MTBF which is the same way that the original failure time was chosen. The new life span is added to the old failure time to obtain the new failure time.

$$\text{new failure time} = \text{old failure time} + randexp(MTBF) \tag{5}$$

Next the component is designated as functional once more and is shipped back to its replace level to be held as a spare until needed. This is done by setting the condition state variable to "functional" and generating a ship event to the depot where the component was originally replaced. Finally, the depot checks its queue to see if there are any jobs waiting. If there are jobs waiting, then an event is generated corresponding to the entity waiting at the front of the queue.

2.2.2 Replace

Table 9: Replace function summary

Variable name	Value
type	replace
who	platform ID number
time	TIME + event duration
cost	event duration * labor rate
amount	N/A

```
t = randexp(MTTR);
GenEvent('replace', 5 , TIME + t, t*laborRate );
```

This function is called when a depot goes to replace a broken or obsolete component on a platform. The duration is determined using an exponential distribution with mean equal to the MTTR of the location where the event will take place. The cost of the job is calculated based on the labor rate of the location. Each of these values is determined when the event is scheduled and is saved for post processing. (figure 25) The function takes the following steps.

1. Find the component in need of replacement

   The function locates the first component that is in need of replacement, is designated to be replaced at this location, and for which a suitable spare can be found. This is done by looping through the list of components on the platform and checking the condition and obsolete state variables. If either the condition is "broken" or obsolete = 1, then the simulation searches the list of spares at the current location and checks for a component with a matching type. If no such component exists then skip to step 4.

2. Replace the component

   In this step the component is diagnosed to identify the nature of the failure. The simulation models this behavior by assigning a "cannot duplicate rate" or "CND rate". In general this is an estimation of the false alarm rate of failures for a particular component type, however in order to simplify the problem for this study the CND rate was assumed to be 0% though the behavior was modeled as a concession to the research sponsors at NAVAIR. If the failure is successfully diagnosed the broken component is replaced. The broken component is removed from the platform by setting its "super" state variable to itself and also removing it from the list of components on the platform. Then the previously designated suitable spare is added to the platforms components list and its "super" variable is set to the ID of the platform. Otherwise, the event "type" variable is changed to "CND" and the function skips to step 4.

3. Check the broken component

   In this step the depot checks to see if the component is repairable, which is modeled by a percent irreparable rate. This is generally low and from speaking to various subject matter experts it was concluded that 1% was accurate enough. If the component is repairable, it is shipped to its designated repair

location. The designated repair location is determined using the assignDepot function and a ship event is generated. In the case that the component is designated to be repaired at the current location, no ship event is generated, rather the component is added to the back of the current location's job queue. If the component is irreparable then it is discarded by scheduling a discard event. The discard event function simply changes the condition state variable to "discarded" and the location to "trash".

4. Check for more components in need of replacement

   If there are any more broken or obsolete components on the platform, the depot will determine where to send the platform next. If one is found, the function checks three things: whether the component is designated to be replaced here, whether that location is an operator, and if there are suitable spares here. Figure 26 is a logic table of the possible states and how the simulation determines the appropriate action. If the the component is designated to be replaced at the current location then in theory it should stay at the front of the job queue and immediately get reprocessed. However, if there are no more suitable spares then it makes sense to send it to a different depot that has spares should one exist. If no other depots can do the job, then the platform is sent to the back of the queue instead. This follows the logic that by the time it gets to the front of the queue again, the depot should have received a new supply of spares. The one exception is if the current location is an operator. In this case it does not make sense to send the platform to another operator, since this will result in an a change to the force structure. Therefore, if no spares are available, the operator will always send the platform to the back of the job queue.

   If the current location is not the designated replace location then it makes sense to send it to the correct location. However, if the correct location is an operator,

then it does not make sense to require that the operators service platforms that do not belong in their operational force. Due to the nature of the logic of the model, there can only ever be two components in need of replacement on one platform at a time; one broken and one obsolete. This happens when obsolescence occurs when the platform is already in the queue to receive a replacement component. Therefore, the first component to get replaced will always be the broken component. In this case, where the obsolete component will get replaced at the operator, the simulation sends it to the warehouse while it is still obsolete. When an operator requests a backup platform it will receive an obsolete one. However, since the platform's obsolete state variable will still be equal to 1 the ship function will add it to the operator's job queue instead of the operational force. In this way, the obsolete component will be replaced and then the platform will be functional again.

If no more broken or obsolete components are found the replacement action is completed by designating the platform as functional. If the current location is an operator the platform is added directly back into the operational force under the assumption that when the operator is designated to perform maintenance it will only perform maintenance on its own platforms and will not send them anywhere else if it can avoid it. If the current location is a depot, the platform will be sent to the warehouse to be kept as a spare.

5. Schedule the next job

   The final step is to check the queue for the next repair or replace job. If it was determined that the current platform had another broken/obsolete component that could be serviced here, the next job will in fact be a continuation of the current job.

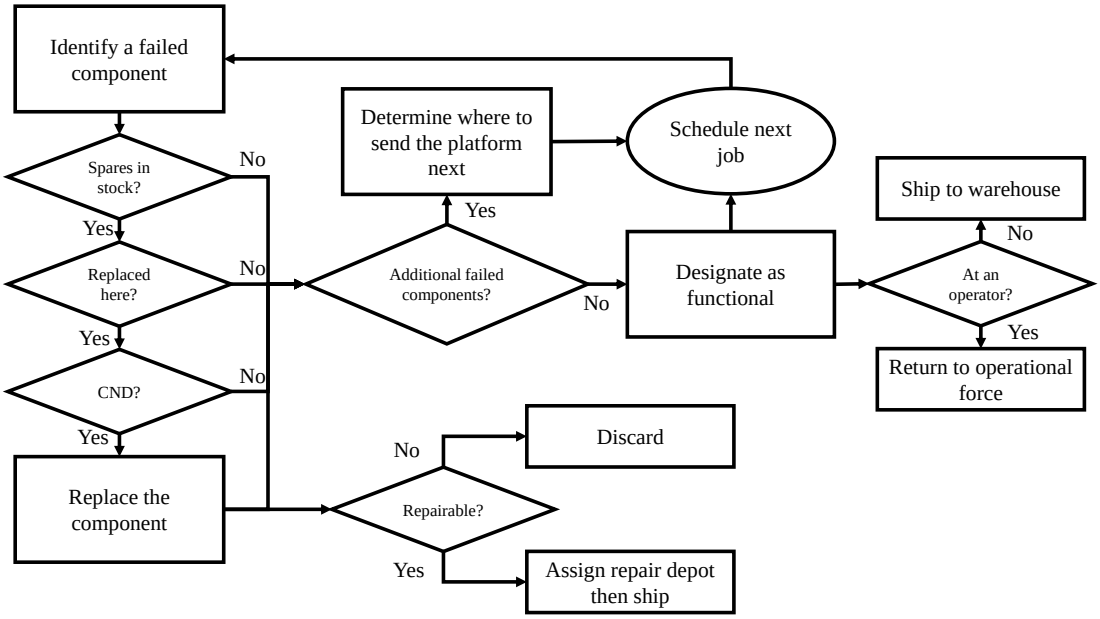


Figure 25: Logic flow diagram for the replace function

	Are there spares in stock here?	Is the component's replace level at the operator?	Is the current location the correct replace level?
Ship to the replace level	false		
Ship to the replace level	true		
Ship to the warehouse			
Ship to the warehouse			
Ship to the replace level			
Stay at the front of the job queue			
Go to the back of the job queue			
Stay at the front of the job queue			

Figure 26: Truth table for the replace function when multiple components on a platform need replacement

2.2.3 Ship

Table 10: Ship function summary

Variable name	Value
type	ship
who	platform or component ID number
time	TIME + shipping duration
cost	shipping duration * shipping rate rate
amount	N/A

```
[cost time] = ShipModel(location, destination, who);
GenEvent('ship', who , TIME + time, cost );
```

The ship function's primary purpose is to oversee the resource transfer between entities in the simulation. Nothing can change location without passing through this function, therefore it is designed to handle many different situations. The function is set up as a series of conditional statements that determine what to do based on the type of entity being shipped, its condition, current location, and destination.

1. If the entity has no components then it must be a component itself. If its condition state variable is "broken" then its destination must be the depot where it will be repaired. In this case it is added to the back of the job queue at that location.

2. If instead the condition state variable is "functional" then it must have just been repaired and its destination will be the depot where it was replaced. In this case it is added to the end of the list of spares at that location.

3. When a component type becomes obsolete, the platform it is on is designated as obsolete as well. This is done by setting its obsolete state variable to 1. The

upgrade function handles where to send the platform next, however the ship function makes sure that the platform is automatically added to the job queue of the location it is arriving at. This is relevant when the obsolete component has been designated to be replaced at the operator to prevent it from accidentally being added back into the operational force. However, if the destination is a warehouse then it is added to the warehouse's "force".

4. Platforms arriving from the warehouse that aren't obsolete will always be going to the operator to resume operations. Upon arrival it is added to the operational force and the simulation checks its components and schedules the next failure event. This is done by finding the component with the least remaining life and determining when that component will fail given the current operational frequency at the operator. Once determined a failure event is generated and added to the event list.

$$\text{time of next failure} = min[\frac{\text{time until next failure} - \text{current age}}{\text{operational frequency}}] \qquad (6)$$

5. Platforms arriving at the warehouse have always just had a part replaced and are leaving a depot. The warehouse checks its wait list and if there is an operator waiting for this particular type of platform it will ship it right away. The warehouse uses its "queue" vector to hold the waitlist information. In this case, the queue is an nx2 matrix where the first column is a list of operators that are waiting for a replacement and the second column is the type of platform that the corresponding operators are waiting for. If the current platform's type matches any of the types in the second column of the waitlist then a ship event will be generated to send the platform to the first operator waiting for that platform type. If no operators are waiting for that platform type then it is added to the warehouse's supply instead.

6. A Platform leaving the operator is always in need of a replacement component and its destination will always be the appropriate replace depot. Upon arrival it will be added to the job queue at that location.

   As a note, whenever an item arrives at a location and is added to the job queue there, the simulation checks for any other jobs in the queue. If there are none, the simulation immediately schedules the arriving item as the next job. (figure 27)

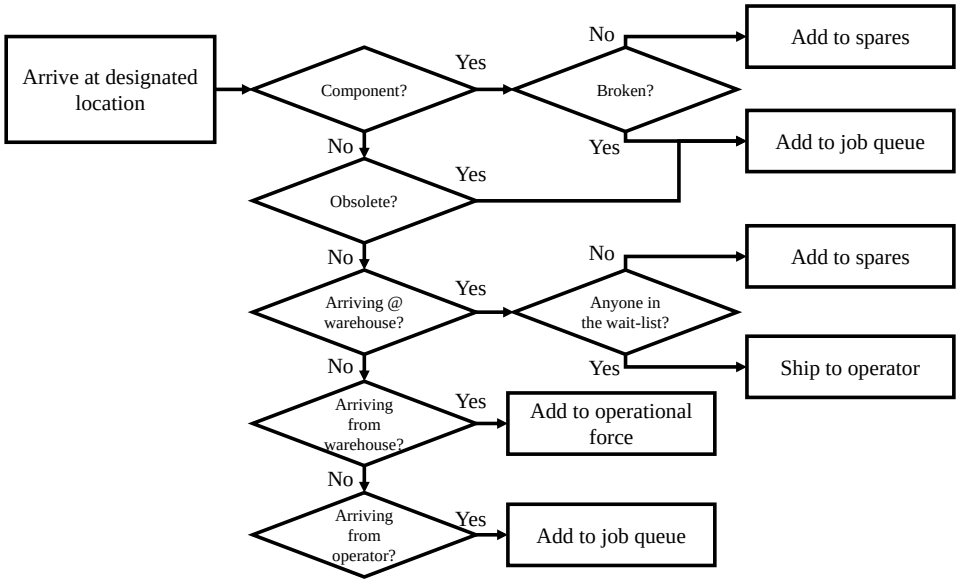


Figure 27: Logic flow diagram for the ship function

A short literature review must accompany the formulation of the shipping cost and duration model. The goal here is not to provide an exhaustive survey of shipping models but rather to pick one model that works adequately. The first step is to understand the components of shipping costs which can be taken from a description of supply chain logistics. The following categories were identified.

- Packing

- Shipping
- Documentation
- Inventory
- Warehouse management
- Transportation planning
- Tracking and delivery

[56] Of those only packing and shipping are relevant for the shipping model, since it is assumed that the shipping support costs are negligible compared to the other costs being considered in this simulation. As such, Baumol and Vinod [23] identify optimal shipping as a trade-off between the following attributes for each shipping option.

- Shipping cost per unit
- Shipping time/speed
- Dependability (variance in ship time)
- Carrying cost per unit time

Aggregation of the costs would logically follow as such.

$$\text{total shipping cost} = \text{fixed cost per unit} + \text{shipping duration} * \text{shipping rate} \quad (7)$$

From the discussion above, it can be assumed that the fixed cost represents the cost of the packaging and the shipping rate is the cost associated with the chosen mode of transportation. For this study, the variance in shipping time is negligible since assuming air transportation, the variance will be on the order of hours. However, the simulation's smallest increment of time is in days, therefore the effects of such small variance will not greatly impact the results.[23] Assuming air

travel and since the motivating problem was originally associated with the maintenance of naval mission computers it can be assumed that the items being shipped are of an order of magnitude equivalent to the AYK-14 mission computer which is the current standard for the F-18. It is between 36 and 45 pounds and has dimensions 14x10.13x7.63 inches. [156].

Additionally, there is a regulation that when shipping electrical components non-conductive packaging must be used since the components are electrostatic discharge sensitive [8]. Depending on the size and weight of the item in question, different qualities of packaging should be used. For example, the mission computer itself weighing up to 70 lbs would require a heavy duty box while the component circuit cards could safely be shipped in a padded mailer. [152]. A quick market search was done to obtain representative values for these costs [121, 151].

Table 11: Packaging costs

Item	value
cost per foot of 24 inch wide antistatic bubble wrap	$.2067
amount of bubble wrap per mission computer	16 inches
cost for a 30x20x20 inch heavy duty cardboard box	$5.82
cost for a padded antistatic box	$4.25

As for transit time and shipping, the model will once again assume air travel exclusively and it will be tuned to the case of naval mission computers. It is assumed that the typical navy transport jet, the C9B skytrainII, is used. From this an average cruise velocity can be found which is used along with the distance traveled to determine the transit time. An assumption is made that it will usually take about one day extra for processing the package on either end of the transport. Next a typical cargo load and the cost per flying hour of the particular aircraft can be found to determine the cost per flying hour per pound of cargo. This is then applied to the weight of the item being shipped to determine the approximate shipping rate

of the item. [41]

Table 12: Air cargo shipping parameters

Item	Value
assumed average air speed	500 mph
cargo capacity	60,000 lbs
cost per flight hour	$8100 FY14

$$\text{shipping duration} = \text{processing time} + \text{distance/speed} \tag{8}$$

$$\text{shipping cost} = \text{packaging cost} + (\text{shipping duration})(\frac{\text{item weight}}{\text{cargo capacity}})(\text{cost per flight hour}) \tag{9}$$

2.2.4 Failure

Table 13: Failure function summary

Variable name	Value
type	failure
who	component ID number
time	-
cost	0
amount	N/A

```
GenEvent( failure , 3, 150, 0);
```

The simulation uses an exponential distribution model of part failure. Each component type is assigned an MTBF during initialization. At the same time the functional life time of each component is calculated by sampling the exponential distribution

with mean equal to the component type's MTBF using the MATLAB randexp function.

When a platform arrives at an operator to begin operations, the simulation checks all the components that make up that platform and finds the component with the least remaining life.

$$T_{failure} = T_{lifespan} - T_{operated} \tag{10}$$

A failure event is then scheduled for only the part that will fail the soonest. This is in contrast to scheduling the failure of all components in the platform and then resetting them when one fails. In this case a failure event would be generated for each component on every operational platform. When one of those components fails the simulation would have to search the entire event list for the remaining failure events corresponding to the other components on the same platform, and cancel them. If the events are not removed then at some later point the simulation would try to fail a component that is not even at an operator anymore. Even worse would be that the component has already returned to an operator and has a second failure event scheduled. When the second failure event triggers, the platform will be asked to be in two places at once and the simulation will crash. However, to remove the extraneous events the simulation has to search the entire event list, which is already large due to all the extraneous failure events for every other platform. This is a computationally expensive task therefore by only scheduling the soonest failure event, the number of items in the event vector is minimized, thereby increasing the computational efficiency of the simulation. When a component fails its state is first set to "broken", then a depot is assigned to receive the platform and replace the broken component. If the part is designated to be replaced at the operator level, the platform stays at its current location and is put at the end of the job queue, otherwise, it is shipped to the proper location. Either way it is removed from the operational force of the operator and the operational life counter is updated for each of the components and the platform itself.

This is done by checking the component's "inopsince" state variable and determining how long the component was operational for since the last time its life counter was updated. Since that is the total duration of time in hours, it must be multiplied by the operational frequency of the operator to determine how long it was actually running for in days.

$$life = life + (TIME - inopsince)\frac{fhperyr}{365} \tag{11}$$

If the platform was shipped to another depot, the operator requests a working platform from the warehouse to replace the one with a broken part. Preference is given to the geographically closest warehouse in order to minimize the wait time. However, if there are no suitable spares at any warehouse then the operator is placed on a wait-list to receive a replacement when one becomes available. In this scenario preference is given to the warehouse with the shortest wait-list. (figure 28)

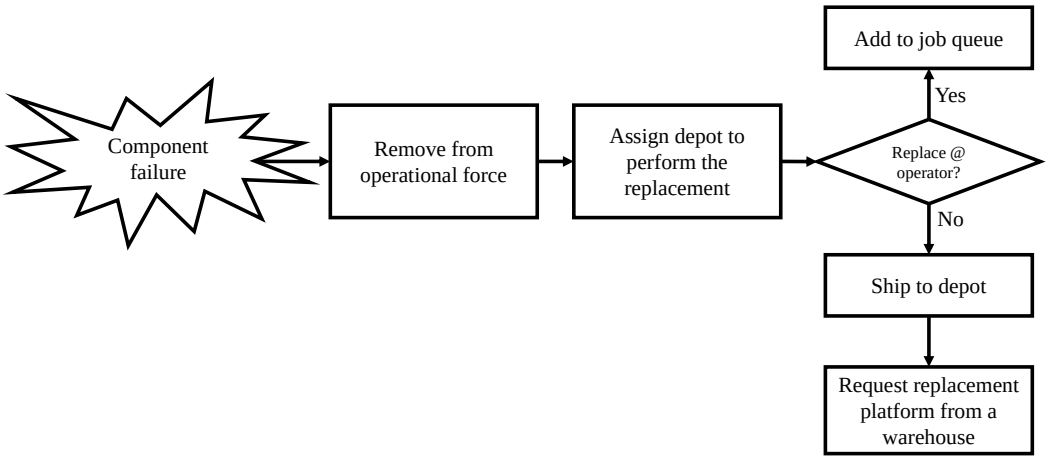


Figure 28: Logic flow diagram for the failure function

2.2.5 Assign depot

Table 14: Assign depot function summary

Variable name	Value
action	“replace” “repair” or “storage”
who	platform or component ID number
depot	location ID number

```
depot = assignDepot(who, action);
```

This function is called whenever an operator or depot wants to ship a part or platform. The function takes as inputs the item being shipped and the mode that the assignment should consider and returns the location where the item should be shipped to. The three modes are as follows;

1. Replace

   When an operator is looking to send a failed platform to receive a replacement part it assigns a depot based on the required replacement level of the broken part. The assignment considers the number of jobs waiting in the queue at each possible location, and the number of spare parts of the correct type in stock as well. Each location is assigned a score equal to the length of its queue, but if it has no suitable spares then that value is incremented by an arbitrarily large (10000 for example). The function will assign the platform to the location with the lowest score. In this way, the assigned depot will be the one with the shortest queue and available spares. If no spares exist at any of the suitable locations then the function will still choose the location with the shortest queue. In this way jobs are presumably spread evenly between all the available depots and turn around times are minimized. Another scenario when this function may be called, is when a platform is already at the depot waiting to receive a replacement part but there are no spares remaining of the correct type. In

this case the depot will check the other available locations and try to send the platform to one of those instead.

2. Repair

   After the broken part is removed from the platform the depot will assign a new location that can handle the repair job. Once again the function will attempt to minimize the wait time by checking the queues of all feasible locations, with the exception that if the repair level matches the replace level of that particular part, the part will be designated to stay in place at the current location in order to reduce shipping costs.

3. Storage

   Finally, once a working part is placed in the platform, the depot will assign a warehouse to store the now functional platform at. The function will favor the warehouse that has the lowest stock of available platforms.

2.2.6 Obsolete

Table 15: Obsolete function summary

Variable name	Value
type	obsolete
who	component type, 0 for random
time	-
cost	0
amount	N/A

```
E = {0,  obsolete , 150, 0, 0};
```

or

```
GenEvent( obsolete ,   C1   , 150, 0);
```

At this point, the discussion of the normal operations of the system ends and the discussion of its disruptions begins. The first disruption to the system is component obsolescence. At this point the condition state variables of all components of the given type are set to "obsolete", the obsolete state variables of all the associated platforms are set to 1, and an upgrade event is generated. It is assumed that the obsolete components will continue to be used until there is a suitable upgrade for lack of a better alternative, therefore, the upgrade event is always scheduled to be the next event. This is done by generating an event with an arbitrarily small duration (.001 for example). It is theoretically possible to remove this assumption and model a design process time between obsolescence and upgrading the component. However, this only makes sense if one also adds a model for the change in behavior during this interim period. For example, the cost of repairing the component could go up as could the cost of procuring new components of that type to correspond to obsolescence due to the original manufacturer discontinuing the product line. For this thesis the simulation is not set up to handle this assumption. Instead a conservative assumption is made that the system has planned in advance for the eventuality of obsolescence and the engineering design process for the upgrade has already been completed such that all that is left is to put the new components in place. Alternatively, this behavior also matches the assumption that an already existing commercial component will be selected as the upgrade naturally reducing the design process time.

2.2.7 Upgrade

This function handles what the upgraded component's characteristics will be, how the system handles the obsolete components and how the new components will be added to the appropriate platforms. The function completes the following steps.

- Determine the new component's characteristics

It can be assumed that there would be an attempt to match the fit, form and function of the obsolete component when choosing an upgrade, therefore all the characteristics will vary according to random distributions centered around the original values. The distributions used and the characteristics varied depends on the problem being analyzed. In practice, it might even be such that upgrading one component may trigger obsolescence in other components, however, that behavior is outside the scope of this thesis and is not considered here. For this thesis the MTBF can increase by a factor of two, and weight and unit cost will all vary by up to 50% in either direction. Additionally the maintenance requirements will change by varying the designated repair and replace levels for the particular component. In this way the network is forced to evolve in order to accommodate the new locations that the platform must go to.

Table 16: Component upgrade model

Component attribute	Upgrade multiplier
MTBF	1+rand(0-1)
Unit cost	.5 + rand(0-1)
Weight	.5 + rand(0-1)
Maintenance plan	randint(1-6)

- Handling the remaining obsolete components

  Obsolete components that are not on a platform are discarded. This is under the assumption that the system will not try to reuse or re-purpose obsolete components that are still functional. If for a future study this assumption is deemed invalid then this behavior must be modified. Additionally, components that are not currently functional and are awaiting repairs should always be discarded and removed from the job queue, since it can be assumed that it would be a waste of resources to repair an obsolete item.

- Restock new components

For each depot that will now be servicing the upgraded component, the simulation will trigger a block buy event to restock the supply of spares at each location. A block buy event is usually triggered after a replacement takes place and there are one or fewer spares of a given type left. In this scenario the block buy function will generate a number of new components equal to the difference in the location's current supply and its ideal stock. Those components are immediately added to the location's "spares" vector.

Alternatively, the block buy function can be used to add additional platforms which will be assigned to a random operator. In this scenario an entire new platform is generated, complete with functional components, and added directly into the operational force of the designated operator. This is used to increase the force structure during the runtime of the simulation.

Table 17: Block buy function summary

Variable name	Value
type	blockbuy
who	component or platform type
time	-
cost	total unit cost
amount	nspares - current stock or externally determined amount

Example external event for a new platform

```
E = {  P1  ,  blockbuy  , 150, 1, 0};
```

or

Example event generation for a set of new components

```
GenEvent( blockbuy ,   C3   , 150, 5);
```

- Upgrading platforms

  When replacing the obsolete components that are on platforms, there are number of scenarios that must be considered. As mentioned before there are three different methods for implementing the upgrades and within each one there must be different protocols for handling the platforms at an operator and at a warehouse. In all scenarios the platforms that are in the job queue of one of the depots remain there and will receive the upgrade when its turn in the queue comes up.

  The first and simplest scenario is upgrade by attrition. In this case nothing special is done, rather each platform will receive the upgraded component at some later point in time when it is already at a depot in need of a different replacement component. This is the least disruptive scenario as it allows for a large spread in which to implement the upgrade. However, because it will frequently require the platform to visit multiple depots before being fully functional once more, this scenario will likely result in slightly worse performance over time especially when multiple upgrades are in process at the same time. Finally, in practice, it is often impossible to use this method due to the criticality of the upgrade.

  In the case of a highly critical upgrade, the system can opt for an immediate field modification team to replace all the obsolete components. In this scenario, all platforms at an operator or a warehouse will immediately receive an upgraded component on site without having to travel to a depot. The simulation will accomplish this by creating the new component at the time of the replacement so as not to impose on the spares supply of the depots. As with any time a platform is pulled from the operational force, the simulation pauses the failure time for that platform and when the upgrade is complete, the failure event is reset. The result is a short dip in availability and a constant upgrade implementation time regardless of the network topology before the upgrade. Due to this fact, this method is the ill suited for measuring flexibility.

Table 18: Field mod function summary

Variable name	Value
type	fieldMod
who	platform ID number
time	TIME + shipping time
cost	component cost + shipping cost
amount	N/A

```
GenEvent( fieldMod , 7 , 150, 5000);
```

Finally, the round robin method of implementation ships all effected platforms to the designated depots for maintenance action. For the platforms at an operator, the operator will be added to the wait-list at the warehouse to receive upgraded platforms when they becomes available. The exception is when the upgraded part is designated to be replaced at the operator. In this case only the platforms at the operator receive the upgrade now, and the ones at the warehouse will receive the upgrade when they arrive at the operator at a later point in time. Once again this makes sense under the assumption that operators are only capable of providing maintenance to the few platforms that they are supposed to be operating.

2.2.8 Depot failure

Table 19: Depot failure function summary

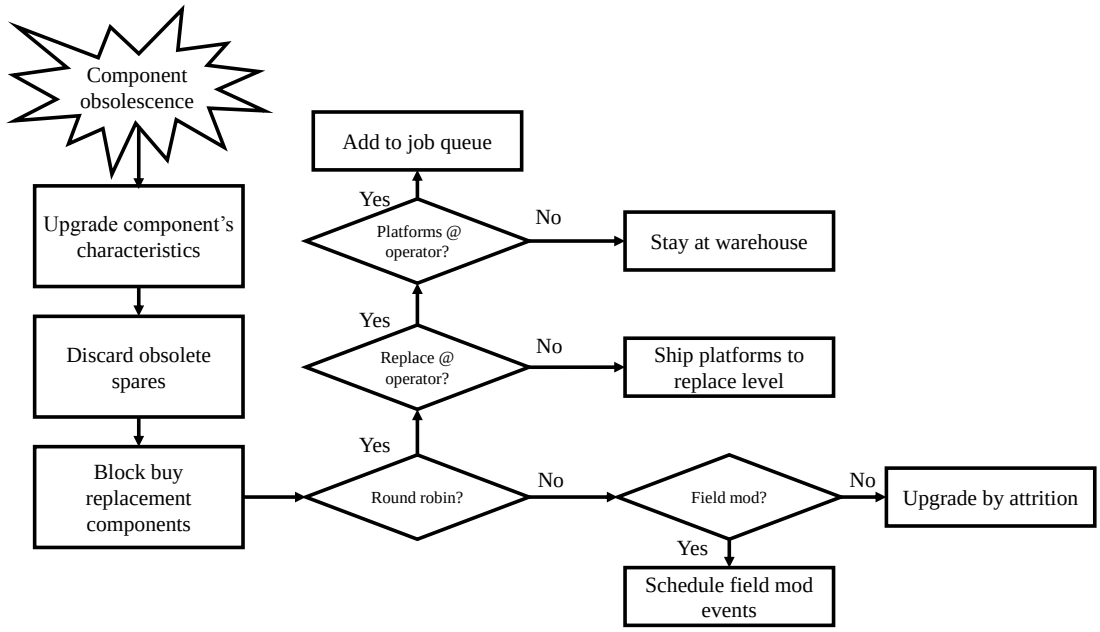


Figure 29: Logic flow diagram for the upgrade function

Variable name	Value
type	failDepot
who	location ID number, 0 for random
time	-
cost	0
amount	N/A

```
GenEvent( failDepot , 4 , 150, 0);
```

The second type of disruptive event is the failure of a depot. It is assumed for the purposes of this study that warehouses and operators don't fail, because in the event that they would, the availability would immediately drop below the acceptable threshold regardless of the design and the results would not be meaningful. For example, if there are two operators each with six platforms. If either operator fails then the number of operational platforms will immediately decrease by half. While

an accurate representation of the behavior of the system, it is also not a meaningful representation of the flexibility of the system, since the goal of modeling location failures is to determine how many locations the system can lose before the job queues start to back up. As such it also doesn't make sense to model warehouse failures, since a warehouse failure will automatically result in an eventual deficit in spare platforms even if there is no back log at any of the maintenance depots.

To summarize, when a depot fails its condition state variable is set to "failed" and the assign depot function will no longer assign items to be serviced there. The result being if enough depots fail, the system will eventually run out of functional platforms as they will be stuck waiting for service at the few remaining depots and the availability will drop. In theory this will favor architectures where most of the maintenance gets done at locations with high redundancy. For example if there are three depots and only one intermediate level of repair, then designating most of the components to be serviced at the depots will result in a more flexible system.

2.2.9 Operational shift

Table 20: Operational shift function summary

Variable name	Value
type	operational_shift
who	N/A
time	-
cost	0
amount	Magnitude of the change in flight hours per year

Example external event for a new platform

```
E = {0,  operational_shift , 150, 100, 0};
```

or

```
n = GenEvent( operational_shift , 0, 150, 0);
EVENTS(n).amount = 100;
```

The final type of disruptive event is a shift in the operational frequency which is accomplished by increasing the flight hours per year of each operator by a designated amount. This is intended to correspond to an increase in demand on the system. It is assumed that if the platforms are utilized more often they will fail sooner, and the more frequently the platforms fail the more maintenance jobs will need to be processed. The expected behavior is that at some point the system will be unable to process jobs fast enough and the performance will decrease. It should be noted that since the failure times of each platform are already set in advance when the operational frequency is changed the simulation must reset the timing of these events to correspond to the change. This is done by finding the next failure event in the event list for each operational platform and canceling it. Then because the operational frequency at each operator was changed the time when that component should fail will also change and a failure event is generated according to the logic documented in section 2.2.3. Since the event list must be searched multiple times this function will require a greater computational cost.

2.3 Choice of variables

In order to sufficiently explore the behavior of the problem a representative subset of design variables that drive the trade offs that are of primary interest will be chosen. In this section an argument will be made for why each design variable is either varied or defaulted to a constant value.

The model inputs can be broken down into four categories.

1. Component properties

- Mean Time Between Failures (MTBF) - the rate at which each component fails
- Unit cost - the cost of procuring a new component
- Weight - component net weight
- Maintenance option - a number between 1 and 6 corresponding to the alternative combinations of repair and replace level for each component

2. Depot properties

   - Mean Time To Repair (MTTR) - the rate at which the depot can process maintenance jobs
   - Labor rate - the cost per day of labor performed at the depot
   - Location - the xy coordinates of the depot
   - Number of spares - the number of spare components that each location tries to keep in stock

3. Operator properties

   - Operational frequency - the rate at which the platforms are utilized in flight hours per year
   - Force structure - the number of each platform that is operated at each location (this is also defined for the warehouse)

4. Platform properties

   The only independent variable associated with each platform is the number and type of each component that it requires.

The primary design variable considered by traditional LoRA is the maintenance requirements of each component. The model make the following assumptions.

- All components that require maintenance at a given level can go to any location of that level. For example, if a component gets repaired at the intermediate level, then it can go to any intermediate level to get repaired. Alternatively, the model is setup to handle a distinction between the sets of components that each maintenance location can service. This is handled by a vector associated with each location named "partsServiced" which is checked by the AssignDepot function. However, when a component is upgraded, this distinction is lost. The Upgrade function will determine the new maintenance requirements and automatically add the component to the partsServiced list of all locations of that maintenance level without any consideration of where that component used to be serviced.

- A repair action must be performed at the same level or higher than the replace was performed. For example, if the replace is supposed to be performed at the intermediate level then the repair can only be performed at the intermediate or depot levels. As such, with three levels of maintenance, there are only six possible combinations of repair and replace levels.

    1. replace and repair at the operator
    2. replace at the operator and repair at an intermediate level of repair
    3. replace at operator and repair at a depot
    4. replace and repair at an intermediate level of repair
    5. replace at an intermediate level of repair and repair at a depot
    6. replace and repair at a depot

    As a note the operator is designated as level 1 and is therefore defined as the lowest level of maintenance in this document.

- Discard always happens at the same level as replace. In some models it is assumed that discard happens after the repair location has failed to repair the component. In this model it is assumed that the replace level has the diagnostic capability to determine whether it is worth sending the component to the repair level in the first place.

This design variable is defined for each component type separately and represents the primary trade off between the fast turn around time but high cost of maintenance at the operator and the low cost but slower option of maintenance at the depot level. The correct choice is likely to depend on the second major component property; MTBF. If a component has a low MTBF then it is likely to fail more frequently. In this case it would make sense to service the component at a lower level since it is more likely that several of that component will be in need of maintenance at the same time and a faster turn around time will be desirable. Alternatively, the components with the lower failure rates (high MTBF) will likely opt for maintenance at the highest levels since the effects of slow turn around times will be less pronounced and the lower maintenance costs will be desirable.

The other two component properties, unit cost and weight, will be defaulted for this problem. The reason for defaulting unit cost is that it is only considered when new components are purchased to determine the cost associated with the block buy event. As such it has no effect on the performance of the system and only the cost, therefore it does not contribute to a meaningful trade off and need not be varied. As for component weight, if the operational performance of the aircraft were a consideration then the net weight would be a consideration. However, since this is not the case, weight is only considered when determining the shipping cost of the component. Just like unit cost this will only effect the total cost of the system and need not be varied. Of the depot properties, only MTTR will be varied, which is defined individually for each operator, intermediate and depot. The reason for this is that MTTR is the

design variable that determines the maintenance turn around time and as mentioned earlier varying turn around time results in a meaningful trade off. Location is not varied since it will only effect the shipping cost and time, but since shipping times are expected to be very small in comparison to the life of the system, small variations in location are not expected to matter very much. Additionally, it can be assumed that the system will attempt to use only existing maintenance assets which is also a reason why the total number of maintenance locations is not varied. Finally, labor rate is not varied for the same reason as mentioned earlier regarding component unit costs. It can also be assumed that if both MTTR and labor rate are allowed to vary then it would be possible to choose a design where the operators have the fastest turn around times and lowest maintenance costs, making them the ideal choice for all components, which does not make sense.

A note about the number of spares. While LoRA can be used to determine the correct number of spares to purchase for each component type, it is more common to use a heuristic model. [126] For this model, it was assumed that the initial stock of supplies be equal to at least 10% of the total number of each component in the operational force. In practice, the simulation will trigger a block buy event to resupply the stock of spares in the case that any location runs out. As such the initial allotment of spares does not significantly effect the performance of the system, only the overall cost, therefore it is not considered as a significant design variable.

In addition to the maintenance properties, each operator is also characterized by the operational setup at that location. The first design variable is the operational frequency of the operator. As described in the section about the failure behavior of the model, the flight hours per year of each operator will effect the failure rate of the components operated there. Similar to MTBF, this will result in a meaningful trade off between cost and performance. However, it effects all components equally and will not result in a change in maintenance plan for individual components rather

it is likely to effect the maintenance choices for all components. For example, if the operational frequency is high it is likely that the best options will be to reduce the maintenance levels for all components in order to reduce the turn around times across the board to correspond to over all higher failure rates.

The second design variable associated with the operators is the force structure of platforms in operation. In theory the overall failure rate of components at an operator is also dependent on the number of platforms operated there, therefore, increasing the force structure would also tend to have a similar effect to increasing the operational frequency. However, it will be shown in the next section that for purpose of measuring the maximum capacity of the system, increasing force structure does not produce meaningful results, therefore only operational frequency will be varied. Alternatively, the capability of the system can be defined as providing maintenance for a given force of aircraft. As such, varying the force structure would be outside the scope of this design problem.

Finally, the platform architectures are not varied, since without a model for the performance requirements of each platform, the choice of components would be arbitrary. While the trade offs between component commonality would be meaningful in theory a second model would be needed to define the compatibility relationships between components and platforms. Additionally, this is a purely discrete design variable, but unlike the maintenance plans for each individual component, there are an infinite number of possibilities without a model to define the constraints. This would make the design space impossible to define, therefore it will be assumed that LoRA can only be done once the platforms in need of maintenance are defined. This is not inconsistent with varying component properties since it can be assumed that one of the purposes of LoRA is to trade off between a set of options for each component type to determine which one to actually purchase.

In summary, four design variables were chosen to represent the design space.

1. Component MTBF
2. Component maintenance requirements
3. Operator flight hours per year
4. Depot MTTR

2.4 Simulated responses

2.4.1 Support/Maintenance Cost

System resources are modeled by the operations and service cost of the system. This is evaluated whenever a item is replaced, repaired, shipped or purchased.

- Repair & replace cost - section 2.2.1
- Shipping cost - section 2.2.3
- Purchase cost - section 2.2.7

When the simulation terminates, the list of events that occurred is saved and the costs of all ship, repair, replace and block buy events are summed to determine the total cost. As such the initial setup cost of the system is not evaluated. It should be noted that upgrade events do not have a unique cost associated with them instead they are characterized by a series of replace events and block buy events.

2.4.2 Availability

There are multiple alternatives for system performance metrics, however only one metric is necessary to show meaningful tradeoffs. For this scenario, availability will be evaluated as it is generally considered the primary performance metric for maintenance systems. Availability is defined as "the average fraction of the time during

which the system is able to perform its function" [46]. For this scenario each platform is considered "available" as long as it is at an operator and in its operational force, therefore the long term availability (a_n) would be the % of time that a given platform is operating.

$$a_n = \frac{1}{\Delta t} \int_{t_i}^{t_f} a_k(t) dt \tag{12}$$

However, the goal is to consider the average availability over the entire system. The obvious method for measuring this would be to evaluate the individual availability of each platform and average over all of them.

$$A = \frac{1}{n} \sum_{k=1}^{n} [\frac{1}{\Delta t} \int_{t_i}^{t_f} a_k(t) dt] \tag{13}$$

Switching the order of integrals is mathematically valid. Conceptually, this changes the formulation to averaging the fraction of platforms that are operating at a given time, which is much easier to implement computationally. Given discrete time intervals, equation 13 becomes

$$A = \frac{1}{\Delta t} \sum A_k \Delta t_k \tag{14}$$

where A_k is the number of platforms that are currently operating divided by the number of platforms that are supposed to be operating at time t_k. To simulate this, the synchronizer will assess the momentary availability for every event and save it for post processing. A_k is the sum of the lengths of all operator force vectors divided by the sums of all operator force vectors saved in the "operator" global variable structure. When the simulation terminates equation 14 is used to evaluate the overall average availability of the system.

2.4.3 Growth flexibility

As discussed in chapter 1, growth flexibility is the ease or cost of changing the system and is a measure of the cost of fixing the system once it has already failed. The generic approach for this would be to take a failure mode of the system and model

the process of reversing it. For this scenario the primary failure mode would be a back log of maintenance jobs caused by either too few depots or too high a component failure rate. In both cases the logical solution would be to increase the number of functional depots, either by fixing broken ones or building new ones until the desired level of availability (ideally %100) is reached. As mentioned earlier modeling the cost of setting up depots is outside the scope of this particular design problem, however in future applications of this methodology such models may exist, in which case the author encourages that they be used to measure growth flexibility.

In the meantime, there is one other disruption to the system that can be simulated; obsolescence. When a component becomes obsolete, all platforms that use that component must receive an upgrade. This causes a dip in the availability until all the platforms are restored. Measuring the duration of this dip can be equated to the ease by which the change was administered. It is recommended that a single upgrade method be used to simulate this response, since allowing for a random choice between the methods will greatly increase the variance in the response. For example, when the field mod method is used the duration of the disruption is always the same regardless of the topology of the network since all the upgrades happen at the current location and all happen simultaneously. Alternatively, when replace by attrition is chosen, there is no dip at all, since the platforms are not ever recalled. While there may be a slight reduction in performance for a period of time, it is hard to distinguish it from the natural behavior of the system. Therefore, round robin upgrade is the logical choice of methods for this formulation.

To simulate this one component, chosen randomly, is set to become obsolete at the start of the simulation. This is done before initializing the simulation, by externally designating an obsolete event with type = 0 to specify a random component type and time = .001. The result will be that the first event in the simulation will be the obsolescence of one component triggering an immediate upgrade event.

```
E = {0,  obsolete , .001, 0, 0}
```

Since the series of replace events caused by the upgrade will cause an immediate dip in availability, the time when availability returns to %100 will be the time when all the replacement events are completed. Therefore, after the simulation has terminated naturally, the time when availability returned to optimal is found in the event history as the measure of growth flexibility.

It must be noted that the alternative of causing all components to become obsolete in sequence and take the average of the response times was not used. While it would likely reduce the variance in the measurement, it would also be very difficult to evaluate due to the possibility of multiple response times overlapping with one another.

Growth flexibility is likely to be primarily related to the average MTTR of all maintenance locations. Since, upgrading will require the replacement of all obsolete components, and will usually change the replace level of the component, the original maintenance requirements will not effect the growth flexibility. Rather, a system with overall low maintenance times at all locations will respond faster on average.

2.4.4 Volume flexibility

Volume flexibility or system capacity flexibility is defined as how much surplus resources are in the system. This is relevant for when the actual demand exceeds the level that the system was designed to handle. In this case the system is considered flexible if it can handle the increase in demand. Quantitatively speaking the degree of flexibility is measured by how much more demand it can handle. As discussed in chapter 1, demand is defined by the number of platforms requiring maintenance at a given moment in time, and the system's capacity is defined by how many of those platforms can be processed at once. Therefore, in order to measure the flexibility of the system, one would have to increase the demand until the system fails. In general,

failure occurs when there are not enough spare platforms at the warehouse to replace the ones that are awaiting maintenance at a depot. The easiest way to identify this behavior is to determine if the current availability is unusually low. In order to distinguish between the typical drop in availability caused by a few component failures and a severe back log of jobs a threshold is defined under which the system is considered failed. For this study %65 was determined to satisfy that condition.

To increase the demand there are two options; increasing the number of platforms in the system, or increasing the rate at which they fail. Increases in force structure are accomplished by defining block buy events. For this study it was set up such that one random new platform would be added to the force per month for ten years.

```
for ii = 1:120
    E = [E; {0, blockbuy', ii*30, 1, 0}];
End
```

Increases in operational frequency are accomplished by defining operational shift events. For this study it was set up such that the operational frequency would increase by 25 flight hours per year once a month for ten years.

```
for ii = 1:120
    E = [E; {0, 'operational_shift', ii*30, 25, 0}];
end
```

The difference between these two scenarios is that if the force structure is increased and all the platforms fail at the same time then the back log at the depots will be

greater in severity, while if the operational frequency is simply increased the the back logs will happen more frequently but will be smaller in severity. In practice, both scenarios were implemented, but increasing force structure did not frequently produce meaningful results. The reason for this was that the definition of system failure involves a minimum threshold on availability, and adding more platforms tends to increase the overall availability. A simple illustrative example would be if there were two platforms and one fails. This reduces the availability to %50. However, if a third platform is added then the availability will immediately increase to %66. Therefore, the more platforms that are added to the system, the larger the back log must be before the availability threshold is hit. The result was that the failure condition was frequently never hit before the simulation terminated naturally and a meaningful measure of volume flexibility would not be found. As such, increases in operational frequency was used to simulate volume flexibility. In practice this was modeled by incrementally increasing the flight hours per year at each operator until availability dropped below %65 at which point the simulation terminates and the total increase is assessed.

The advantage of not increasing force structure is that increasing the number of platforms also increases the length of the vector of entities and the vector of events, resulting in significantly greater computational costs. In conclusion, both behaviors are consistent with the increase in demand that would accompany a shift from peace-time operations to war-time, a capability which is expected from a Navy maintenance system, however one is easier to implement computationally.

2.4.5 Divisibility

Divisibility has to do with the robustness of the paths by which the system can provide the desired capability. In this scenario, the paths are all viable routes that take a platform from the operator with a failed component through the maintenance cycle

back into operations. To simulate a disruption to the system for this purpose, one must break the paths until the system fails. There are two ways to describe a path; by links or by nodes. It can be inferred from the previous section that the connections between depots cannot fail, since it is unlikely that a connection dependent on air travel will be disrupted for more than a day, whereas the depots can fail for significant periods of time. Therefore, to measure divisibility, the depot failure function is used to disable nodes until the availability threshold of %65 is met and the number of such events is noted as the measure of divisibility. For this study it was set up such that one random depot would fail at one year intervals using the following code to set up the external event vector.

```
for ii = 1:nDepots
    External = [External; {0,  failDepot , ii*365, 0, 0}]
end
```

This is not an entirely accurate evaluation since this will always favor scenarios where most of the components are maintained at the operator level. However, as mentioned previously this assumption must be made. In general though, this will result in a system with high divisibility but also very high costs. Alternatively, a system with multiple depots that service most of the components will also be fairly divisible due to the high degree of redundancy, but will likely have lower costs than the previous example. An example of a system with low divisibility would be one where all the components must pass through a single intermediate level of maintenance. When that location fails the availability will quickly drop soon after.

2.5 Stochasticity of the responses

Since this is a stochastic simulation, some effort must be expended to account for the variance in the response. Generally this is accomplished by running repetitions of each case and determining the mean of the responses. To determine the correct number of samples, a statistical P-test was used to find how many repetitions must be run in order to place the mean of the response within a given interval with 95% confidence. For this thesis getting the mean within 5% of the true value was determined to be sufficient therefore the following equation was used to determine the sample size.

$$n = \frac{16\sigma^2}{(\mu/10)^2} \tag{15}$$

Where σ^2 and μ are the true variance and mean of the response. In this case where the true values are unknown they are the variance and mean of a very large sample size which in this case was 10000 repetitions of the example problem. Since, the scenarios used to simulate each of the measures of flexibility are different, their required sample sizes must be evaluated independently. The results can be summarized as follows for the baseline system architecture.

1. cost & availability - 1 repetition
2. divisibility - 72 repetitions
3. volume flexibility - 159 repetitions
4. growth flexibility - 489 repetitions

This is not entirely unexpected. For cost and availability the response is the aggregation of thousands of random events, therefore it makes sense that the variance and the required sample size would be small. However, small sample sizes are susceptible to the effects of outliers, therefore a conservative sample of 100 repetitions was used, since the variance on the response of other architectures may be higher.

Also, when disruptions are considered, it is possible that the system will only fail a fraction of the time and a single simulation run cannot accurately capture this effect. This also eliminates the effects of model crashes. With regards to growth flexibility it was mentioned that the choice of formulation would likely increase the variance (see section 2.4.3), therefore this result is also not unexpected. In conclusion, It seems that evaluating flexibility would be computationally more expensive than evaluating cost and performance. Therefore, in the case that suitable indicators for flexibility are found but not for performance and cost, a reduction in computational cost by a factor of over 500 can still be realized by using heuristics for flexibility only.

2.6 Model validation

In order to validate the simulation it is necessary to compare it to real data. In this case the Deptartment of the Navy PMA209 conducted a study titled Advanced mission computer and displays (AMC&D) life cycle cost estimate that used a similar model, the joint aviation model (JAM). [118] It was essentially a multi-platform LoRA analysis looking exclusively at the mission computer related components in the F-18 C,D,E, and F with the goal of making a business case for replacing the older AN/AYK-14 mission computers. [4]. The study considered the four main components of the mission computer.

1. Mission Processor (MP)
2. Display Processor (DP)
3. Display Head 5"x5" (DH5x5)
4. Display Head 8"x10" (DH8x10)

The distinction between the two sizes of display heads is that the larger one is exclusively used in the twin seat F-18F super hornet. [156] Up until this point the information is all publicly available, however the exact quantities of each component is

not and cannot be published in this thesis. Suffice it to say the validation experiment was set up to mirror the AMC&D scenario as closely as possible. The goal was to run FLORA DES and attempt to match NAVAIR's value for service costs per aircraft per year while maintaining an acceptable performance level. The variables that were copied included the following:

- Mission computer architecture
- Maintenance strategy
- MTBF
- Weight
- Unit cost
- Maximum life span
- Number of operators (aircraft carriers)
- Number of spares
- Distance between operator and depot
- Operational rate
- Number of depots

What is missing from the above list is mean time to perform maintenance, labor rates, and the transportation assumptions. With regards to shipping, the assumptions stated earlier in the chapter are used. However, for maintenance costs, an initial run using the values listed in the NAVAIR study resulted in a very poor approximation. The depots were getting backed up and the availability was dropping below acceptable levels. It was presumed that there must be a difference between the way JAM treats

maintenance actions and the way FLORA DES does. FLORA DES considers repair and replace actions to be separate where as it is assumed that JAM treats them as one event. The result was that using the NAVAIR values for maintenance time, FLORA DES effectively doubles that value. After making this change results that were on the same order of magnitude as those documented in the study were obtained. Slight modifications to some other values in the model resulted in values for cost per aircraft per year within %10 of the desired values, at which point it was determined that an exact match could be found given enough time and therefore the FLORA-DES model was sufficiently validated.

CHAPTER III

FLEET WIDE LEVEL OF REPAIR ANALYSIS NETWORK MODEL (FLORA NET)

The FLoRA Net model is a network model that is intended to represent an abstract view of how the multi-platform maintenance logistics system should behave. The model assumes two types of nodes, agents and tasks. The task chain starts with operations, which is defined for each platform type individually. Second, is replace which is defined for each component type individually. Third, is repair which is paired with replace. Finally, storing the spare platforms is defined as a separate task, as that is where they generally go after being repaired. Then there are four types of agents, however they are similar enough that for the network description they will all populate the same graphs. This is because they are all stationary facilities, however, a distinction will be made when defining the relationships. The first type is the operator which is the only type of agent that possesses the actual aircraft, therefore all failures happen at an operator. Second, are the depots, the final level of maintenance. They are usually responsible for dealing with anything that cannot be maintained elsewhere. The intermediate levels occupy a space between depots and operators, as they are generally used to handle the easy jobs and are usually located closer to the operators in order to reduce shipping time. Finally, the warehouse represent the location where the spare platforms are stored until they are needed. Figure 30 contains the definition of the set of agents and resources in the system. In order to effectively demonstrate the application of the methodology, the number of elements is kept to a minimum.

The following notation will be used when describing the networks

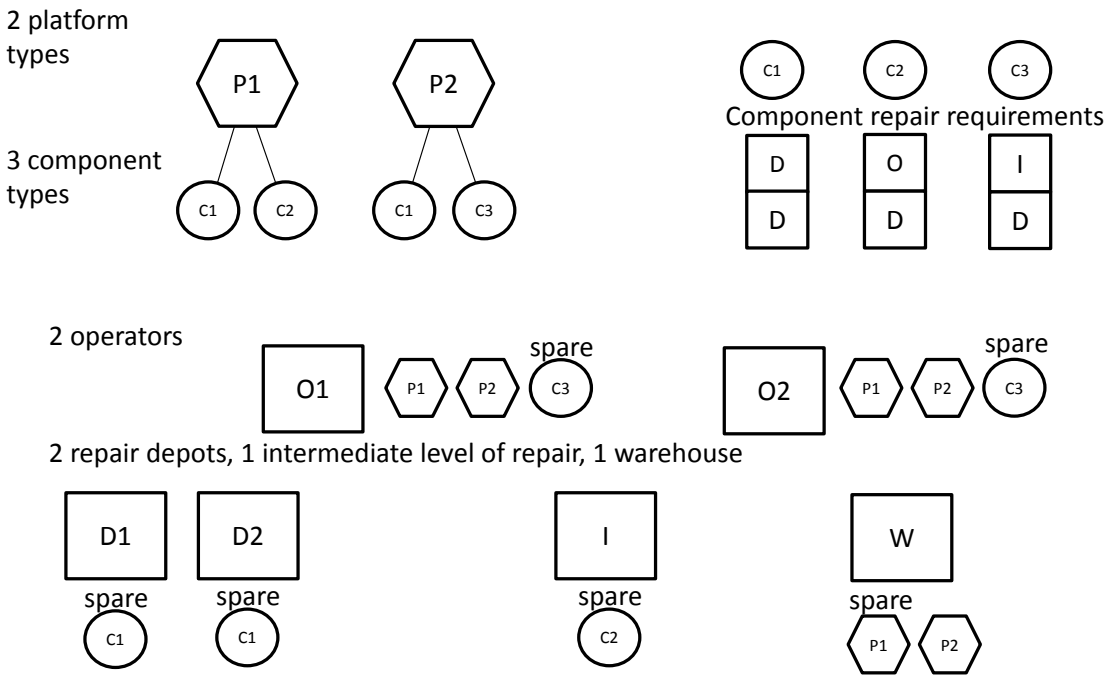


Figure 30: A depiction of the elements that make up the example problem

- T: the set of tasks
- G: the set of agents
- P: the network relating tasks to one another
- P_{ij} an edge weight in the P matrix
- A: the network relating tasks to agents
- A_{tg} an edge weight in the A matrix
- A^d_{ti} the degree of a task in the A matrix
- A^d_{gi} the degree of an agent in the A matrix
- N: the network relating agents to one another
- N_{ij} an edge weight in the N matrix

3.1 Task to task network

Starting with the task to task relationships, they represent the order that events occur. The basic order is that a platform is operational until something breaks. Since, each platform contains different components it is necessary to distinguish between operate tasks. When a component breaks, the platform that it is on is shipped to the maintenance level where it will receive a replacement component. Since each component has different maintenance requirements the replace tasks must be distinguished. Therefore, for each component type on a particular platform there is a precedence relationship between the operate and replace tasks. After the replacement component is placed on the platform, it is sent to the warehouse, therefore links are added from each replace node to the storage node. However, there is an exception, if a component is replaceable at the operator then the platform never has to leave and therefore does not need to go to storage. In this case a link is added returning to the operate node.

After that the broken component must go to be repaired. This is represented by links between the replace and repair nodes for each component type. Additionally, once repaired the component returns to the replace level to be held as a spare, therefore those links are bi-directional. Finally, links are added from storage to each of the operate nodes representing how the operators receive spares when platforms are sent to other locations. Figure 31 demonstrates this network for the example problem.

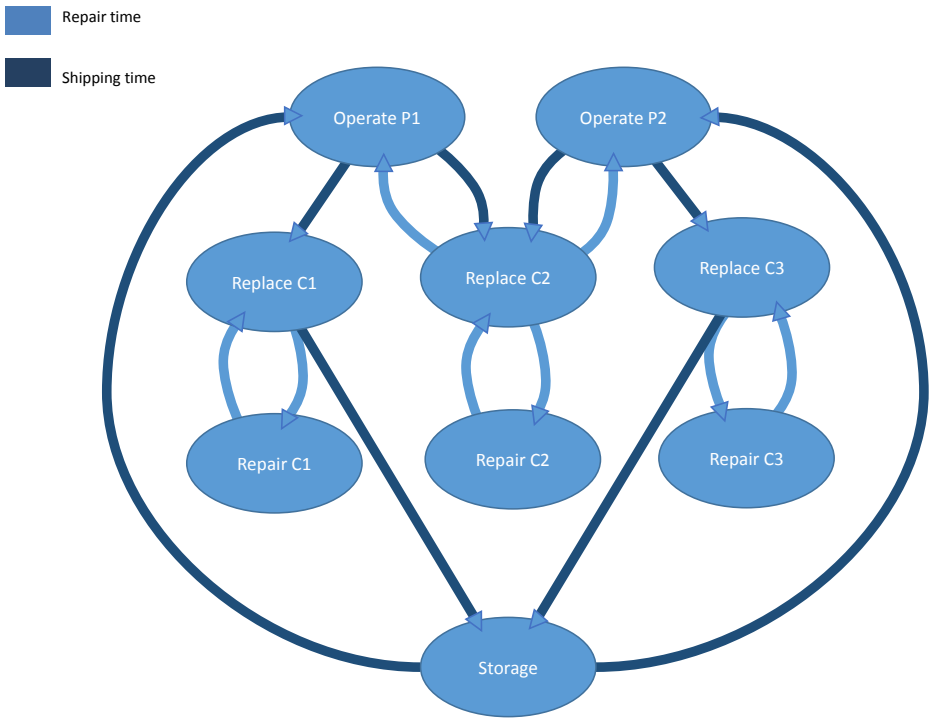


Figure 31: The task to task relationship network for the example problem

With regards to the edge weights, this network is a good candidate for modeling the flow of resources between tasks, therefore the edge weights will represent the capacity of each relationship in terms of the number of items that can pass along that link per unit of time. For the links between operate and replace, the limiting factor on these transitions is only the distance between locations, therefore the capacity of the edge weight P_{ij} will be defined for platform i and component j as the sum of the inverse of

shipping time for all routes that the item can take between n operators and m replace levels. This relationship also applies to the transition between storage and operate.

$$P_{ij} = \sum_{n=A_{ti}^{d}} \sum_{m=A_{tj}^{d}} \frac{1}{Tship_{nm}} \tag{16}$$

For the links between replace and repair, the limiting factor is the turn around time of the maintenance action. Mean time to repair (MTTR) is often used to denote this quantity. Additionally, replace and repair are modeled by a queuing process, therefore the ability of a location to perform a specific task is limited by all the other tasks it is supposed to be doing that use the same queue. In this case the capacity of the edge is the sum over all locations that can perform the task of 1/MTTR divided by the number of other tasks that that are performed at that location.

$$P_{ij} = \sum_{k=A_{ti}^{d}} \frac{1}{(MTTR_i)(A_{gk}^{d})} \tag{17}$$

This relationship applies to all edges leaving the replace and repair nodes. This network best represents what the system does. When the system is working properly there is a steady flow of events that is represented by the transitions in this network. Therefore, a measure of the ability of the system to synchronize its behavior makes sense here, as does a measure of its stability and for this it was previously stated that graph energy and algebraic connectivity were useful indicators.

Graph energy is the sum of the absolute values of the eigenvalues of the adjacency matrix. In MATLAB the "eig" function is used to evaluate the matrixs eigenvalues and eigenvectors.

```
[eigVec1 eigVal1] = eig(PP);
GE = sum(sum(abs(eigVal1)));
```

The algebraic connectivity is the second smallest non-zero eigenvalue of the laplacian matrix. The laplacian matrix is defined as the degree matrix minus the adjacency matrix. The degree matrix will be a diagonal matrix where the non-zero entries will be the degree of each node in the graph which is evaluated by summing the columns and rows of the adjacency matrix with all non-zero entries equal to 1. The eigenvalues are then computed and the second smallest one is found by first finding the smallest one and setting its value to infinity before using the min function a second time.

```
temp = PP;
temp(temp~=0) = 1;
degree_matrix = zeros(size(eigVal1,1));
for ii = 1:size(eigVal1,1)
    degree_matrix(ii, ii) = sum(temp(ii,:)) + sum(temp(:,ii));
end
laplacian_matrix = degree_matrix - temp;
[eigVec2 eigVal2] = eig(laplacian_matrix);
temp = max(eigVal2);
temp = temp(temp ~= 0);
temp(temp == min(temp)) = inf;
AC = abs(min(temp));
```

Additionally, this network contains a description of how the items go from being broken to repair back to being operational. This is represented by the cycle of operate, replace, repair, storage back to operate. As such functional cyclicity was discussed as an indicator of the number of cycles in the graph which in this case represent the ideal behavior of the system. Functional cyclicity is the largest real eigenvalue of the adjacency matrix. These three spectral measures can be calculated since the matrix

will be square.

```
FC = max(max(eigVal1(imag(eigVal1) == 0)));
```

Alternatively, since the edges are modeled as the capacity of the state transition in terms of events per unit time, the maximum flow of this network from operate to storage can be evaluated to represent the maximum rate at which the system can process maintenance requests. This is then compared to the rate at which each platform is likely to fail to determine a measure of the excess capacity of the system, which is used to evaluate volume flexibility. To evaluate maximum flow, the matlab_bgl graph theory package is used. For all operate tasks the maximum flow is evaluated using the max_flow function. The volume flexibility for that task is the percentage of excess capacity that exists which is one minus the load divided by the flow, where the load is equal to the aggregate failure rate of all platforms of that type. Finally the overall system volume flexibility is the assumed to be the mean of all operate tasks.

$$\text{Platform MTBF} = \frac{1}{\sum \frac{1}{\text{Component MTBF}}} \tag{18}$$

$$\text{Task load} = \text{Aggregate failure rate} = \frac{\text{N components of type P}}{MTBFofplatformP} \tag{19}$$

$$\text{Task excess capacity} = 1 - \frac{\text{load}}{\text{max flow}} \tag{20}$$

```
temp = [];
```

```
for ii = find(strcmp({tt{:,1}}, 'operate') == 1) % for each operate task
    for jj = find(strcmp({tt{:,1}}, 'storage') == 1) % to each storage task

% find the total number of platforms in operations of type ii
  nn = 0;
        for kk = 1:size(O, 1)
            nn = nn + O{kk,3}(tt{ii,2});
        end
        load = nn/P{tt{ii,2}, 3}; %load = 1/mtbf;



        flow = max_flow(sparse(PP)*10000, ii, jj)/10000; % find max flow
% the factor of 10000 is because the max_flow function rounds the output to the
% nearest whole number.
% This method effectively produces more decimal places in the response.

        if flow > 0 % no negative flow allowed
            temp(end+1) = 1 - load/flow; % evaluate excess capacity
        else
            temp(end+1) = 0;
        end


    end
end
```

To summarize, the four network properties that result from this model are as follows.

1. Graph energy
2. Functional cyclicity
3. Algebraic connectivity
4. Max flow

3.2 Task to agent network

The next network model describes the relationship between agents and tasks. In this case the matrix will not be square making many of the network properties hard to apply. As such, the edge weights for this graph will be zero or one. For each task a link will be added to each agent that can perform that task. This network provides an easy to use mapping of agents to tasks for the purpose of determining redundancy on the task side and overloading on the agent side. The higher the degree of a task node, the more agents there are that can perform it, the higher the redundancy and the more likely it is to get done quickly. Being a measure of the number of options available for that task makes this useful for measuring divisibility, but it may also simply be a generic indicator of all kinds of flexibility. In this case the task with the lowest degree provides a lower limit to the flexibility of the system.

For example, in figure 32 repair C2 has a degree of 2 with either D1 or D2 capable of performing that task. In the case that one of the depots goes down then the task can still be performed. Alternatively, for replace C3 the only available location is the lone intermediate level of maintenance and should that location fail or all the C3s need replacement at the same time due to an upgrade, there will could a significant drop in performance.

On the agent side of the graph, the higher the degree of an agent node, the more tasks it is responsible for, and the more likely it will be to get a backup of maintenance jobs. In this case the agent with the highest degree is an upper limit on the flexibility of the system. For example, D1 and D2 have a degree of four meaning they are responsible for four different tasks. If a backup of jobs would occur it would be logical to look at the most heavily burdened agents first.

The two network properties for this network model are listed below.

1. Minimum degree of all task nodes

2. Maximum degree of all agent nodes

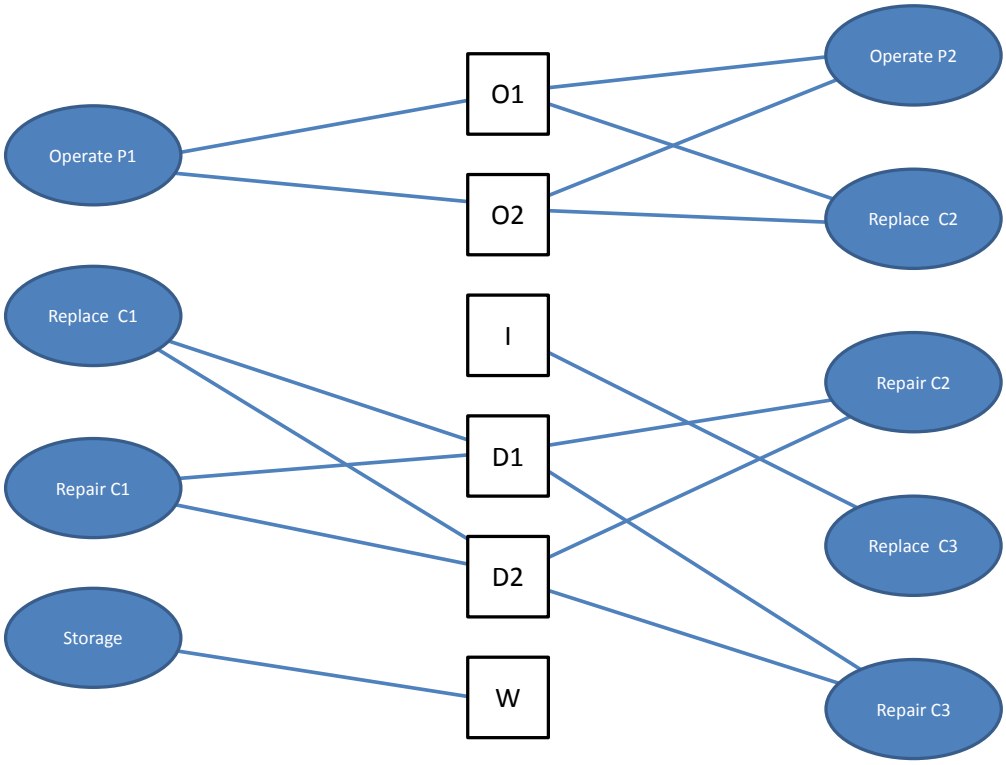


Figure 32: The agent to task relationship network for the example problem

3.3 Agent to agent network

The final network contains the relationships between agents. For this problem, there are some similarities to the task to task network since items tend to change locations when they transition from one task to another. However, the transition capacities are harder to quantify since each link must represent multiple different component types. Therefore, this graph is better suited for showing the loads on the connections between maintenance locations in terms of the number of component types that use that edge.

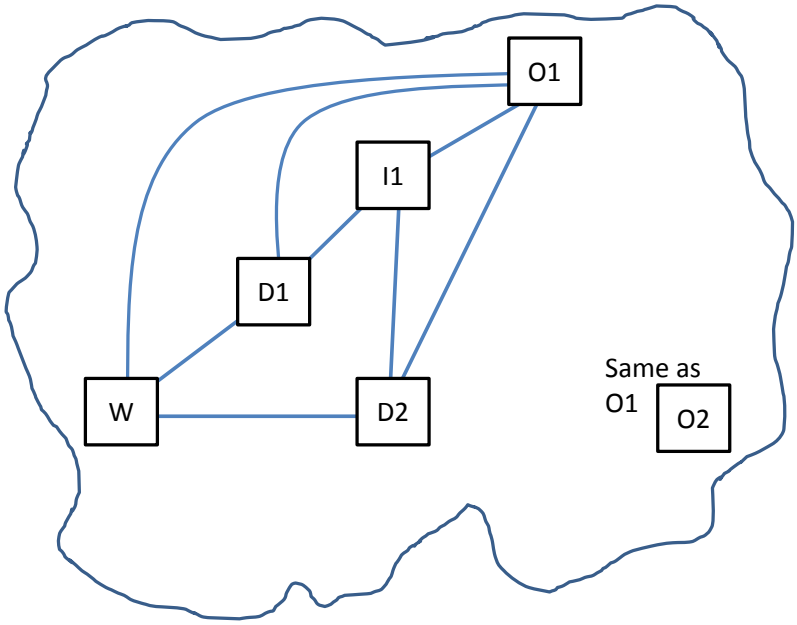


Figure 33: The baseline agent to agent relationship network for the example problem

There are two different ways to look at this network in terms of how to measure flexibility. The first is the expected load on each edge given a certain type of change, which is good for determining the aggregate load on each agent similar to the way the previous model was used. The second way is to model the probability that the edges exist after a given type of change, which is good for counting paths in the network. Additionally, there are two types of changes that can be applied to this network; the removal of nodes or edges and a change in edge weights . The addition of a node does not make sense in this formulation since the expected load on any new depots will be undetermined. As such, there are four sub models that can be derived from this network.

1. The expected value of the load on the edges

   For each edge the number of component types that can travel along it is counted. For example, a component that is repaired and replaced at a depot will travel from the operator to the depot and its platform will continue on to the warehouse, therefore all O-D and D-W edges will be added. If another component is replaced at the operator and repaired a depot, it will also travel along the O-D lines but will return to the operator after being repaired along the D-O lines. In this case all O-D lines would have a weight of 2.

Table 21: Table of edge logic

repair level	replace level	required edges
O	O	no edges
I	I	O-I-W
D	D	O-D-W
O	I	O-I-O
O	D	O-D-O
I	D	O-I-W & I-D-I

This network has two uses. First, similar to the agent to task network, the in-degree of an agent is a measure of the level of responsibility that it has. As before, the agent with the highest degree limits the flexibility of the system. The identification of such a node can be an indicator of a choke point in the network which will result in poor performance in the face of lost nodes.

The second use is to aid in the formulation of the next three models by being the baseline loading structure of the system before changes.

2. The probability of an edge existing after the removal of a node

   The weight of each edge is the probability that it will exist after one depot is removed. In this scenario, it only makes sense to remove depots and intermediate levels of maintenance. If an operator is removed the drop in performance of the system can deterministically be assessed due to the nature of the way availability is measured. When an operator fails, so do all its aircraft at which point the operational force of the entire system drops by the number of aircraft that are stuck at the failed operator. Additionally, the removal of a warehouse will have a deterministic effect on the system as well. Without access to a supply of spares the system will eventually run out of aircraft and the availability will drop to zero. However, depending on the structure of the network, it is possible that the system could continue functioning with fewer depots. In this case each depot (I and D levels) has a probability of failing of $1/n$.

The result is that for every edge that has a load of at least one, the probability of it surviving is $1 - P_{fail}$ which is $1 - 1/n$ for edges incident on only one depot and $1 - 2/n$ for edges incident on two depots.

With a probability assigned to each edge it is then possible to assign a probability to each possible path through the network. Using a modified depth first search algorithm the number of paths from the operator to the warehouse is counted. In general, algorithms for counting paths do not exist since the possibility of getting stuck in a cycle will result in infinite solutions. However, in the case of systems of systems it can be assumed that getting stuck in an infinite loop is not going to actually happen, since the agents involved will realize and set themselves back on the correct path. Therefore a modified depth first search algorithm is proposed for counting paths from all source nodes to all sink nodes in a network. A depth first search decomposes the network connections into a tree and searches down each branch of the tree. The modification that is added is that when a node is repeated the branch is terminated so as not to consider cycles.

(a) Start at the source

(b) End a branch when the sink is reached or when a node is revisited

(c) Explore all possible branches

(d) Count the number of branches that end at the sink

(e) Repeat for all combinations of sources and sinks

Figure 34 demonstrates an example graph where the path finding algorithm identifies six possible paths. There is one cycle in the graph and the algorithm correctly identifies when the path has gotten stuck in an infinite loop. The MATLAB implementation of the path finding DFS can be found in appendix C.

For each path, the probability that it exists is the probability that none of its edges fail.

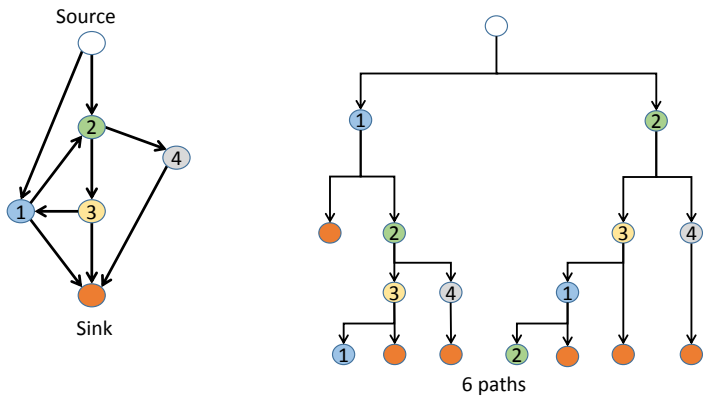


Figure 34: Example execution of the path finding DFS

$$P_{\text{path fail}} = 1 - \prod P_{\text{edge fail}} \tag{21}$$

These values are aggregated for all the possible paths and can be used as measure of divisibility, since in theory as long as at least one path still exists then resources can flow through the network and platforms can still be maintained to some degree.

```
% see appendix c for explanation of PathCountDFS
[nPaths, paths] = PathCountDFS(sparse(NN2), o, w);
NN2paths = ones(nPaths,1);
for ii = 1:nPaths % for each path
    for jj= 1:numel(paths{ii})-1 % for each edge in path ii
      % find the probability that it fails
        NN2paths(ii) = NN2paths(ii)*NN2(paths{ii}(jj), paths{ii}(jj+1));
    end
end
```

Similarly the graph theory concept of connectivity refers to the number of elements that must be removed to disconnect the graph. This is analogous to the number of depots that must fail before there is no longer a viable path for resources to flow along. This is done by assuming a modified definition of connectivity to count the minimum number of elements that must be removed before no possible path remains. This path connectivity problem can be reformulated as a min-cut problem between a source node and a sink node, where all edges have value 1. This way the value of the minimum disconnecting cut will be the total number of edges removed. If the edge weights are probabilities then this will represent the fact that the edges will only be part of the disconnecting set a portion of the time. The path connectivity is evaluated for each operator as a source node and the total system path connectivity is the average value.

```
NN2conn = [];
for ii = o % for each operator
    for jj = w % for each warehouse
            % evaluate path connectivity
NN2conn(end+1) = max_flow(sparse(NN2)*10000, ii, jj)/10000;
    end
end
```

Finally, the criticality of each edge can be determined by the ratio of its load and the number of paths that rely on it to the number of alternative paths that exist that do not use that edge. In this case the most critical edge is an upper limit on the divisibility of the system. The criticality is a measure of the cost of rerouting should a path be disrupted. It would follow then that the measure of criticality would be related to the number of paths that pass through a certain edge, similar to

betweenness. Additionally, if the capacity of the edge and the expected load on it is known then these can be taken into account as well. It will be assumed that when a path is disrupted that the load that was initially intended to pass along it will be redistributed across all other available paths. This is analogous to the scenario where there are two depots and when one of them fail all the jobs that would have originally been split between the two location must now go entirely to the lone remaining one. The criticality of the edge can then be defined as the ratio of the load on the path to the number of alternative paths that do not use that edge. For edge i with load L_i, there is a set of paths P that accomplish the same capability. The number of paths that use edge i is p_i, therefore the number of paths in P that don't use edge i can be called p_i^o and the criticality is $\frac{L_i}{p_i^o}$.

```
A2B = zeros(nPaths, 2); % A2B is a vector that stores the source and sink of eac
for ii = 1:nPaths
    A2B(ii,:) = [paths{ii}(1) paths{ii}(end)];
end

[i j k] = find(sparse(NN2));
NN2pathcrit = zeros(numel(i), 1);
for ii = 1:numel(i) % for each edge
    M = [];
    for jj = 1:nPaths % find paths with edge i
        if ~isempty(strfind([paths{jj}], [i(ii) j(ii)]))
            M(end+1) = jj;
        end
    end

    if ~isempty(M) % if any paths contain edge i
```

```
            altPaths = [];
            for mm = 1:numel(M) % for each of those paths
                    % find alternative paths
                    a = find(A2B(:,1) == paths{M(mm)}(1));
                    altPaths = [altPaths setdiff(a', M)];
                    % paths that start at the same source but end at any sink
                    % and are not in M
            end
            % flex_i = #alt paths/total edge load
            if ~isempty(altPaths)
                NN2pathcrit(ii) = NN1(i(ii), j(ii))/numel(unique(altPaths));
            else
                NN2pathcrit(ii) = 100;
% if no alternative paths then it should be highly critical
            end
        end
end
```

3. The probability of an edge existing after an upgrade causes a slight rearrangement in the network structure

   This network uses similar logic to the previous one, except instead of using depot failures to determine the probabilities, the end state of the network after an upgrade is used. When a component is upgraded, there is a chance that the location where maintenance is performed will change resulting in a change in the loading structure of the network. In this case it is assumed that the repair and replace levels will always change and be randomly reassigned to one of the six options. Using the logic in the table above, the probability that a certain edge will change value can be determined. For this network the probability that the edge will remain after the upgrade is being considered, therefore an edge with a load greater than one will automatically remain

after only one change. For edges with exactly one load, the probability that it loses its load is the probability that the one component that uses that link is the one being upgraded and the probability that the new maintenance option does not use that link. However, for all links that currently have zero load, the probability that they gain a load is the probability that the new maintenance option uses that edge. Evaluation of flexibility for this network is the same as that for the previous network.

$$\begin{aligned} N3_{ij} &= 1 \text{ if } N1_{ij} > 1 \\ N3_{ij} &= 1 - \frac{1 - P_c}{m} \text{ if } N1_{ij} = 1 \\ N3_{ij} &= P_c \text{ if } N1_{ij} = 0 \end{aligned} \tag{22}$$

Table 22: Table of P_c values

Edge type	P_c
D-W	2/6
O-I	3/6
O-D	2/6
I-D	1/6
I-O	1/6
D-I	1/6
D-O	1/6

4. The expected value of the load on the edges after an upgrade

   Using the same logic as the previous network, the expected value of the load on each edge can be determined using the following equation.

$$N4_{ij} = N1_{ij} + \frac{(m - N1_{ij})P_c}{m} - \frac{N1_{ij}(1 - P_c)}{m} \tag{23}$$

   This network is used in a similar way to N1 as a way to find the most heavily loaded location and to be used in conjunction with N3 to determine the criticality of the

edges.

The eight network properties that are defined for these four network models are listed below. Combined with the four properties from the P network and the two from the A network results in a total of fourteen network properties evaluated by the FLoRA NET model.

1. Maximum degree of all nodes in N1
2. Maximum degree of all nodes in N4
3. Total number of paths in N2
4. Total number of paths in N3
5. Path connectivity of N2
6. Path connectivity of N3
7. Maximum path criticality of edges in N2
8. Maximum path criticality of edges in N3

3.4 Chapter summary

In summary, there are three different network descriptions of the system that are used to measure flexibility. The first one is the task to task model and will be used to measure the volume flexibility as the maximum rate that the system can process events and the growth flexibility as the spectral properties of the system. The second network is the agent to task model and is mainly used to help formulate the other two models. However, the degree of the nodes in this graph may provide bounds on the divisibility of the system although it may no be related to just one measure of flexibility. Finally, the third network is the agent to agent relationship model and is used to explore the effects of the different types of changes to the system. The

probability and expected loads on the edges are used to count the paths through the network, determine the path connectivity and identify critical edges as a measure of divisibility with respect to lost nodes and growth flexibility with respect to the rearrangement of connections.

CHAPTER IV

SIMULATION RESULTS

With the two models set up and running it is now possible to start turning the knobs and obtaining results. This chapter will describe how each experiment was conducted and discuss the results and their consequences.

4.1 Model setup

4.1.1 Scenario definition

The baseline scenario that was used to conduct the experiments was as follows. There are three different types of platforms each with three components. However there are only five component types therefore there is some overlap between the platforms. To service these platforms there are two operators, one intermediate, three depots and a single warehouse. Each of the operators has six total platforms though the number of each platform type varies between the two operators. Meanwhile the warehouse keeps two of each platform as spares. As for spare components, each depot will attempt to keep a stock of five spare components of each type. Finally with regards to the component properties unit cost and weight are both kept constant with all components weighing 5 pounds and costing 100 dollars.

To clarify the nomenclature, C1 refers to component type 1. The same logic applies to P for platforms, I for intermediate, O for operator, D for depot, and W for warehouse. The distribution of platforms in the network is as follows.

Table 23: Simulated force structure

	P1	P2	P3
O1	2	2	2
O2	2	1	3
W	2	2	2

The components for each platform are as follows.

Table 24: Platform architectures

Platform	Components		
P1	C1	C2	C3
P2	C1	C4	C5
P3	C2	C4	C5

The maintenance locations are distributed as follows. All distances are in miles relative to the warehouse.

Table 25: Maintenance site locations relative to the warehouse

Location	Coordinates
O1	1500, 1500
O2	0, 2000
D1	500, 500
D2	0, 1000
D3	500, 1500
I	1000, 1000
W	0, 0

The labor rates for each maintenance location are also fixed in units of dollars per day.

Table 26: Labor rates

Location type	Labor rate
Operator	\$50/day
Intermediate	\$30/day
Depot	\$15/day

4.1.2 Design variable ranges

In order to generate alternatives the design variables defined in chapter 2 were assigned values according to uniform random distributions on the ranges in the table below. For the repair and replace location of each component there are only six feasible combinations as was described in chapter 2, therefore those two variables are combined into one variable and assigned a random integer between 1 and 6. For the remaining design variables, continuous random distributions were used.

Table 27: List of design variables and ranges

Variable name	Type	Range
C1 maintenance plan	Integer	[1,6]
C2 maintenance plan	Integer	[1,6]
C3 maintenance plan	Integer	[1,6]
C4 maintenance plan	Integer	[1,6]
C5 maintenance plan	Integer	[1,6]
C1 MTBF	Double	[500, 4000]
C2 MTBF	Double	[500, 4000]
C3 MTBF	Double	[500, 4000]
C4 MTBF	Double	[500, 4000]
C5 MTBF	Double	[500, 4000]

D1 MTTR	Double	[5, 50]
D2 MTTR	Double	[5, 50]
D3 MTTR	Double	[5, 50]
I MTTR	Double	[1, 20]
O1 MTTR	Double	[5, 50]
O2 MTTR	Double	[5, 50]
O1 flight hours per year	Double	[200, 600]
O2 flight hours per year	Double	[200, 600]

4.1.3 Data collection and visualization

4.1.3.1 Primary design space

First, a random sampling of the design space was conducted. 5000 random systems were evaluated by varying the design factors described above. The goal was to obtain a sample that thoroughly explored the solution space defined by a two dimensional plane of cost and availability. This process took several weeks due to the slow nature of the simulation. Even though a typical run of the simulation would run for under a half of a second, the need to run repetitions of each scenario increased the run time by two orders of magnitude. Additionally, since each measure of flexibility requires a unique scenario to simulate, which also require repetitions, this adds another order of magnitude to the run time. Therefore, assuming it takes between about 500 seconds per design, 5000 alternatives would take 2.5 million seconds which is about 28 days. Figure 35 shows how the simulation runtime increases roughly linearly as the total number of simulated platforms is increased. By the time 100 platforms are considered, which is a on the low end for real systems, the computational costs have increased to the point where even this small scale design space exploration would take almost a

year to complete.

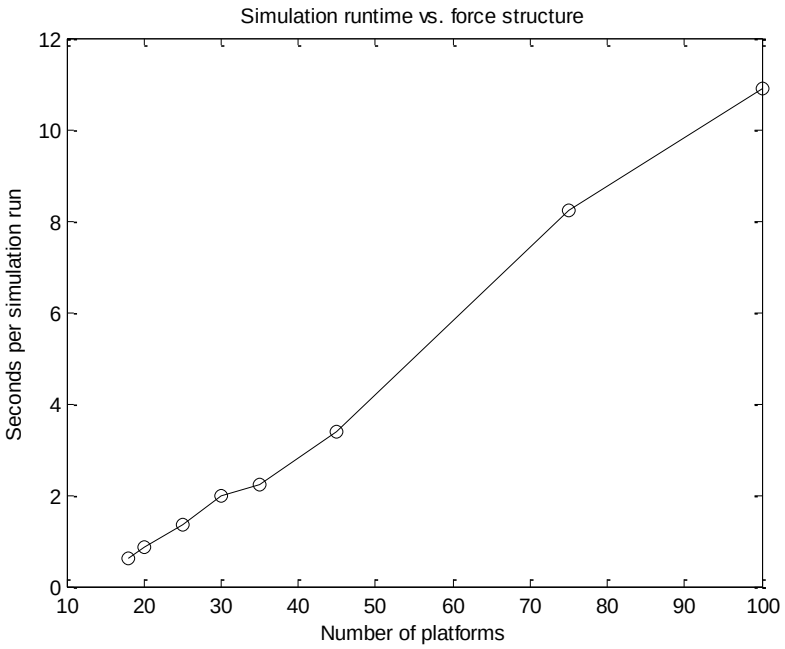


Figure 35: Simulation runtime increases as more platforms are considered

For each design, FLoRA DES was used to evaluate cost, availability, growth flexibility, volume flexibility and divisibility. Additionally, varying levels of disruptions were applied to the design and the cost and availability were assessed for each design. This was done by first randomly applying one of each disruption type to the system then two of each type, and finally three of each type for a total of three, six and nine random disruptions. The code sample below demonstrates how the events are set to happen randomly any time during the simulated ten year life span, and for operational shifts a random value between 1 and 500 flight hours per year is chosen for the magnitude of the shift.

```
E = [{0, 'obsolete', randi(10*365), 0, 0};
     {0, 'failDepot', randi(10*365), 0, 0};
     {0, 'operational_shift', randi(10*365), randi(500), 0}];
```

After that each design is evaluated using FLoRA NET resulting in 14 additional responses. In contrast to the other simulation, a single evaluation of FLoRA NET only takes about 20 milliseconds to complete. Additionally, being a deterministic model, only one evaluation is required per design. In conclusion, the same 5000 designs would only take about 100 seconds to evaluate which is nearly five orders of magnitude faster than FLoRA DES.

4.1.3.2 Example data points

Table 29: Example of one of the best performing designs: system architecture

Variable set	Values
Component maintenance plans	5, 5, 3, 4, 3
Component MTBF	2625, 3920, 2902, 2165, 3025
Average MTBF	3025
Depot MTTR	41, 29, 25
Intermediate MTTR	6
Operator MTTR	6, 7
Operator FH/YR	254, 232

Table 30: Example of one of the best performing designs: simulated performance

Simulated response	Value
Cost	38579
Availability	0.92
Divisibility	1
Growth flexibility	87.73
Volume flexibility	2292.9

The first thing to note about this design is the relatively high values for component MTBF meaning that the cost is generally going to be lower because fewer component failures results in fewer maintenance events that cost money. Similarly, the availability will also tend to be high. However, this says nothing about the flexibility. Examining the maintenance requirements for each component will reveal how well distributed the maintenance jobs will be across the various depots. For this design, four of the components have some kind of maintenance done at the depot. This is favorable since there are three depots to handle all the jobs and two of those depots even have relatively low MTTRs. Additionally, three of the components rely on the two operators for maintenance and only one component uses the single intermediate. This distribution of responsibility is resilient hence the high divisibility. As for volume flexibility, the favorable distribution of responsibility coupled with relatively low MTBFs and some low MTTRs indicates a high maintenance capacity. Finally, with respect to growth flexibility, the handful of maintenance sites with low MTTR favors a quick upgrade response time.

Table 31: Example of one of the worst performing designs: system architecture

Variable set	Values
Component maintenance plans	4, 1, 1, 4, 4
Component MTBF	547, 605, 3595, 1153, 1870, 1554
Average MTBF	1554
Depot MTTR	49, 28, 9
Intermediate MTTR	12
Operator MTTR	40, 42
Operator FH/YR	347, 426

Table 32: Example of one of the worst performing designs: simulated performance

Simulated response	Value
Cost	282000
Availability	0.615
Divisibility	0.181
Growth flexibility	575
Volume flexibility	182.9

In comparison to the best design, this design is significantly worse. The cost is 86% higher and the performance is 33% lower. The first indicator of this poor behavior is that the average MTBF is nearly half of the value for the better design, meaning components are simply failing more often putting greater stress on the system. The second major difference is in the distribution of responsibilities. For this design all five components rely on operators and three of them use the intermediate, meaning that the system is only using half of the available resources. Additionally, the MTTRs at both operators is particularly high, meaning that with all jobs going through the operator there is likely to be a serious back up, resulting in very low volume flexibility.

Finally, in the case that the intermediate fails, nearly all the platforms will be unable to receive maintenance, resulting in low divisibility.

4.1.3.3 Secondary design space

A second design space was evaluated that conforms more closely to how traditional LoRA is done. For this only the maintenance requirements of each of the five components is varied resulting in $5^6 = 7776$ possible designs each of which is evaluated. The results from this design space exploration will be investigated in greater detail in chapter 5. However, in some cases it will be used to further justify conclusions made from the first data set. Below are additional baseline attributes that were used for this evaluation.

Table 33: Defaulted values for component MTBF

Component	MTBF
C1	500
C2	4000
C3	750
C4	1800
C5	2500

Table 34: Defaulted values for location MTTR

Location	MTTR	FH/Yr
O1	15	420
O2	15	300
D1	10	
D2	10	
D3	10	
I	5	

4.1.3.4 Data histograms

The first method for viewing the results is a response histogram (Figure 36) where the distribution of each variable is plotted. For example the peak of the availability without disruptions histogram is around 0.92 meaning that it is the most common value among the 5000 designs.

It can be inferred from here that the presence of disruptions tends to increase the cost and reduce the performance of the system. This is an intuitive result, which lends additional credibility to the behaviors modeled here. Similarly if the histograms for varying levels of disruptions are overlaid upon one another it further reinforces the observation that disruptions increase cost and decrease availability on average. What is interesting however, is that more disruptions tend not to increase cost by as much while availability continues to drop. This could be because cost is dependent on components failing, but if availability is low then there are fewer platforms in operation so there are fewer component failures resulting in less cost increase.

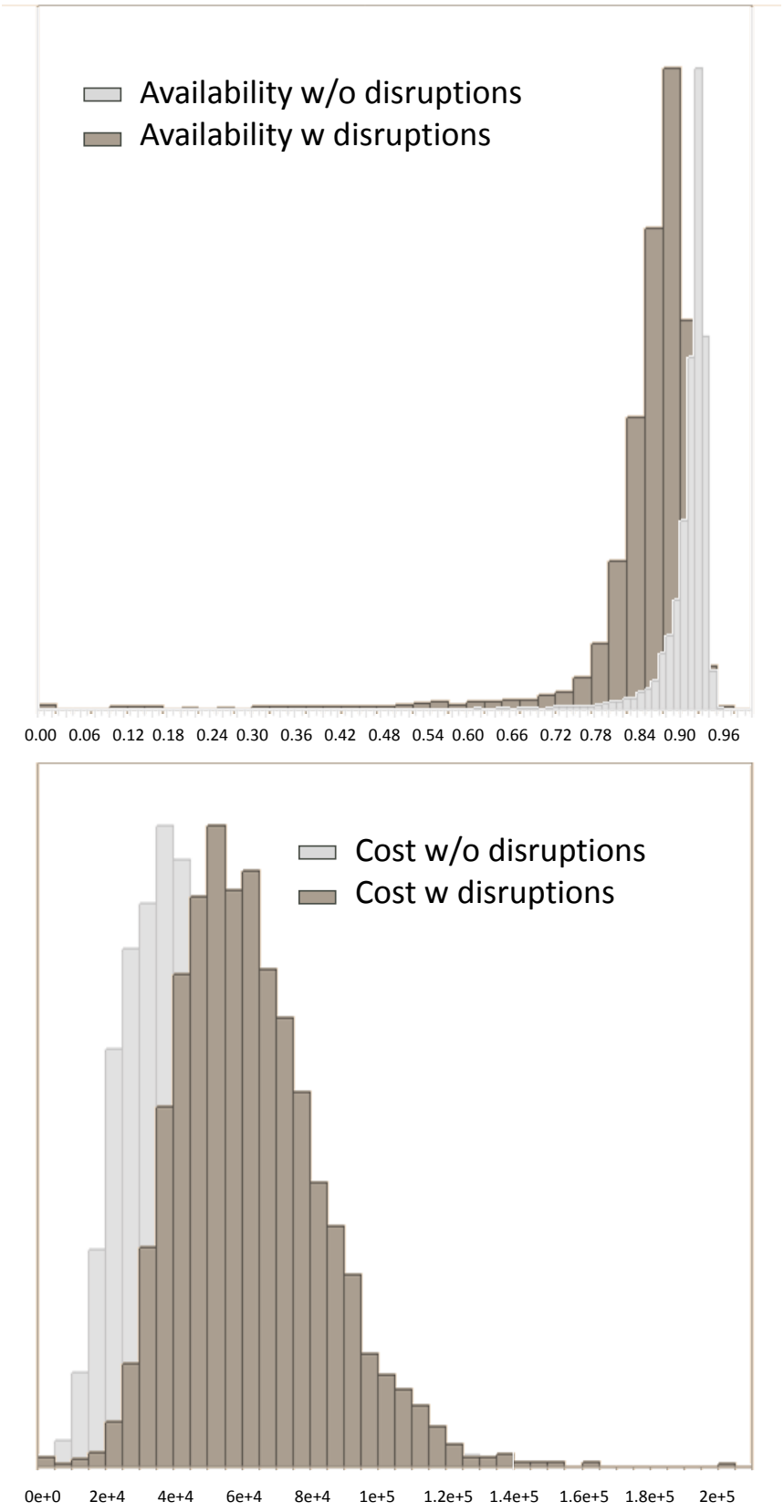


Figure 36: Histograms comparing performance and cost with disruptions and without

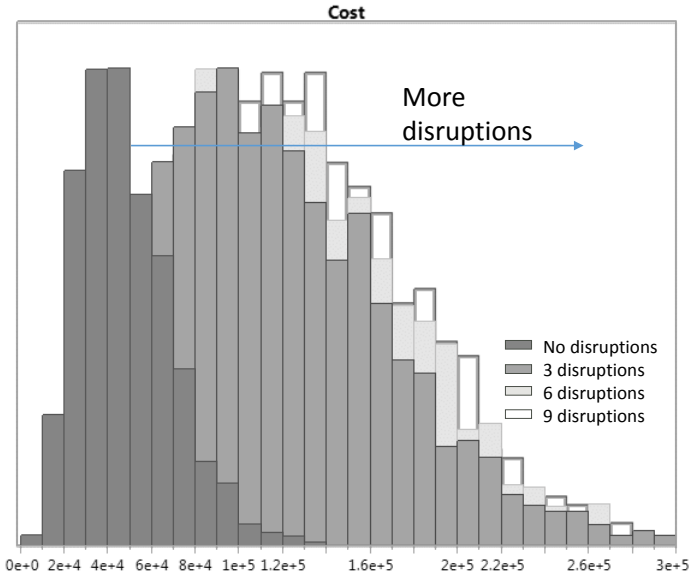


Figure 37: Histograms showing the increase in cost due to disruptions

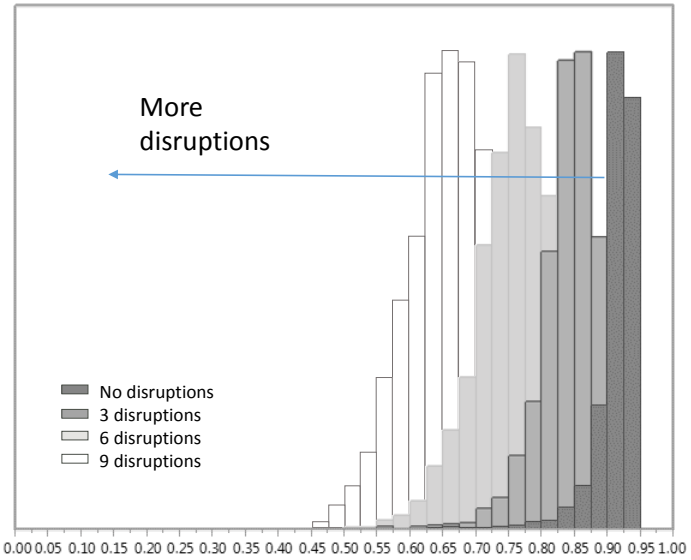


Figure 38: Histograms showing the decrease in availability due to disruptions

4.1.3.5 Scatter plots

From the histograms it might seem like it would be possible to simply quantify the shift in the distributions due to disruptions and plan for the difference. However, this would not take into account whether some designs are shifting more than others. The second method for viewing the aggregate characteristics of the design space is by using scatter plots. For any pair of simulated responses, all 5000 designs can be plotted on a Cartesian coordinate plane, where each point in the plot represents a single design. Doing so allows the reader to view trends in the data. For example the pair of scatter plots in figure 39 show the relationship between the cost and availability of designs before and after disruptions. While there is some correlation between the responses the strength of the relationship is not strong enough that disruptions can simply be ignored. For the purposes of this thesis correlation is defined by Pearson's correlation coefficient which describes how close the relationship between two variables is to being linear. Using similar logic it can be shown that the apparent relationships weaken significantly the more disruptions are considered, and that availability is more sensitive to disruptions than cost is.

Table 35: The effect of increasing the number of disruptions on the correlation to an undisrupted system

Response 1	Response 2	Correlation coefficient
Cost no disruptions	Cost 3 disruptions	0.7866
Cost no disruptions	Cost 6 disruptions	0.7369
Cost no disruptions	Cost 9 disruptions	0.6882
Availability no disruptions	Cost 3 disruptions	0.7682
Availability no disruptions	Cost 6 disruptions	0.6687
Availability no disruptions	Cost 9 disruptions	0.5942

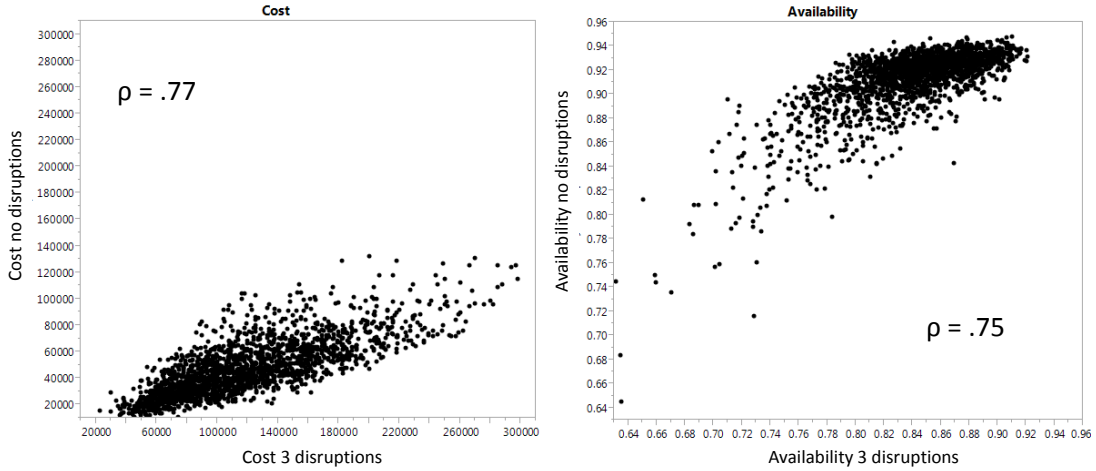


Figure 39: Relationships between availability and cost with disruptions and without

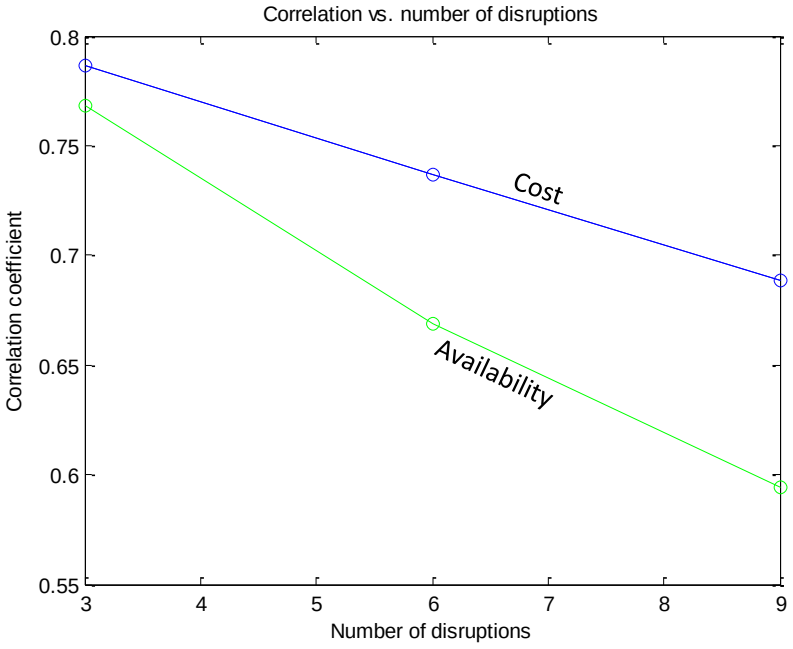


Figure 40: The effect of increasing the number of disruptions on the correlation to an undisrupted system

To further explore this result (figure 41) shows the relationship between cost and availability for the case of no disruptions on the left and three disruptions on the right. The first observation is that for this problem, most of the designs have high availability ($>$ %85) and that cost is highly variable for most values of availability. This is apparent from the slender vertical distribution. However, when disruptions are modeled the distribution spreads significantly. If the lower cost solutions are highlighted then it becomes apparent that some of the designs suffer more due to disruptions than others. To recall one of the hypotheses, flexibility should help identify which designs will fare better in the presence of disruptions.

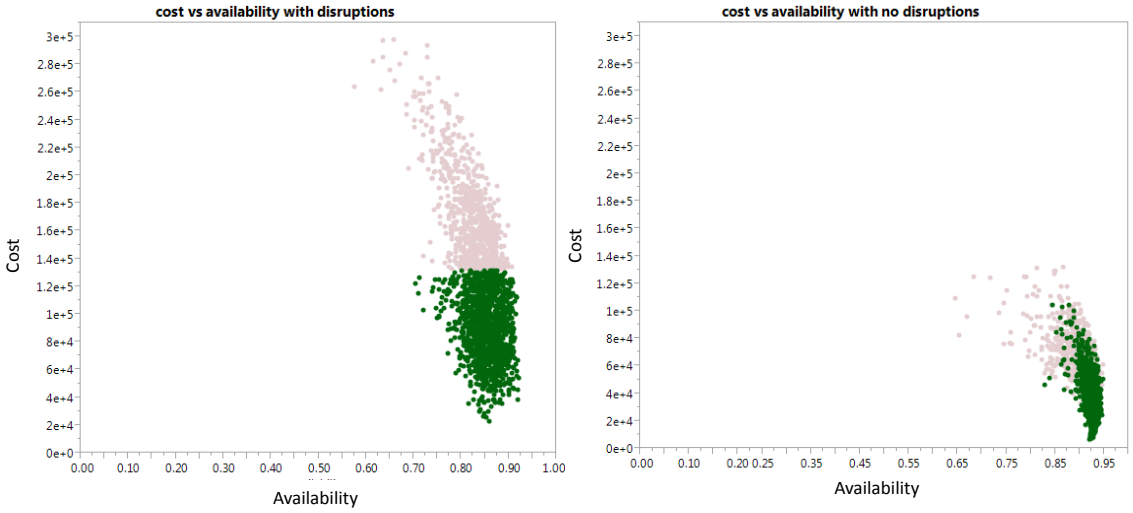


Figure 41: Cost vs. availability with disruptions and without, color coded to identify good solutions

4.2 Correlation between flexibility measures

Between growth flexibility and divisibility a slight negative correlation of -.484 is observed (figure 42). Growth flexibility is measured here as the amount of time it takes for the system to recover from an obsolescence, therefore low values are desirable. For divisibility higher values are desirable since it is measured by the percentage of depots that must go down before the system fails. As such this negative trend indicates that these two measures of flexibility tend to complement one another. However, the correlation is not strong enough to warrant using one measure over the other.

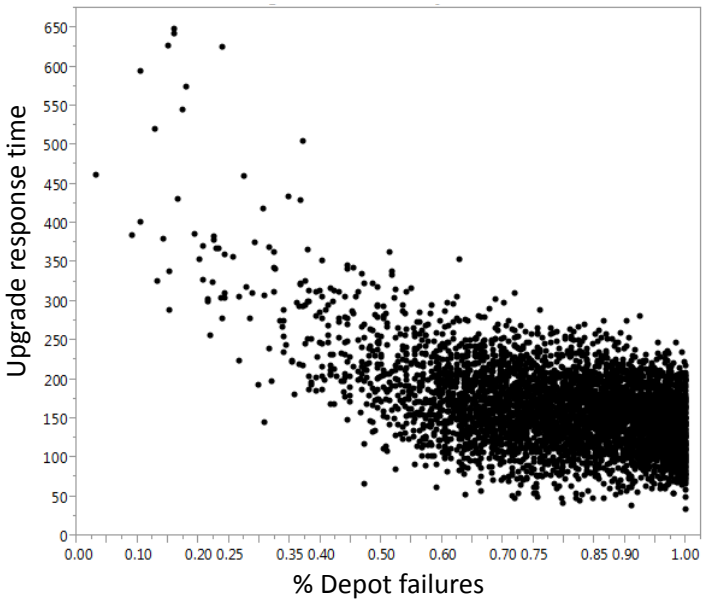


Figure 42: Relationship between growth flexibility and divisibility

For the relationship between divisibility and volume flexibility, a slightly stronger correlation of .5555 is observed (figure 43). Since volume flexibility is measured

as the maximum operational frequency that the maintenance system can support, higher values are desirable. Therefore, these two measures of flexibility are once again complementary to one another. However, the slightly exponential relationship between the responses indicates a diminishing returns effect. For low values of volume flexibility there are large gains in divisibility. However, one the volume flexibility exceeds about 700 flight hours per year, the benefit drops off as divisibility cannot exceed %100. Once again, the correlation is not strong enough to warrant using one measure over the other.

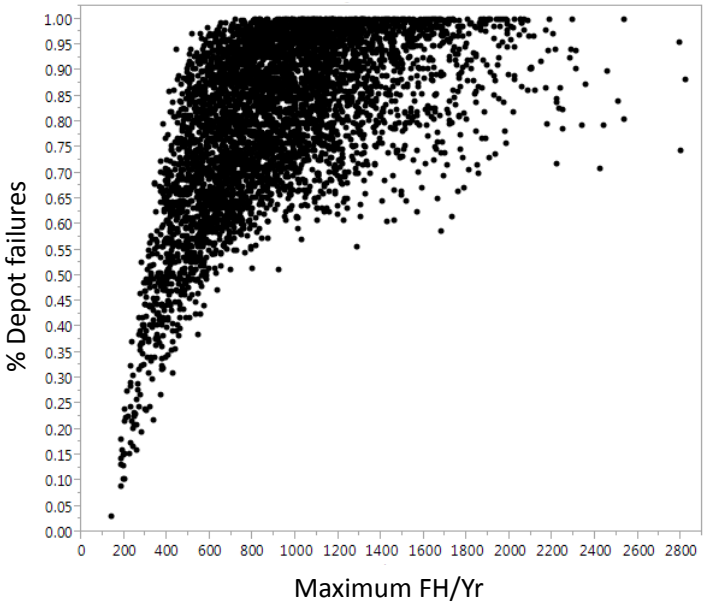


Figure 43: Relationship between volume flexibility and divisibility

The final relationship is observed to have a slight negative correlation of -.5636 (figure 44) . The measures are once again complementary since low values of growth flexibility and high values of volume flexibility are desirable. As with the previous pair, there is also a diminishing returns effect here as well. This is expected given the slight

correlation and exponential relationship between growth flexibility and divisibility.

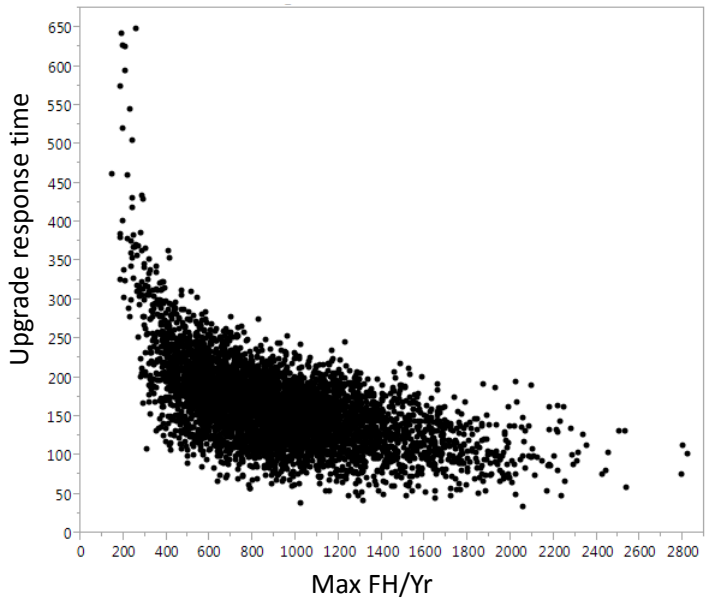


Figure 44: Relationship between growth flexibility and divisibility

In conclusion, all three measures of flexibility are complementary to one another, meaning there are no major tradeoffs. One explanation for this behavior is that systems that can process maintenance jobs faster will inherently be able to respond better to disruptions. However, there is enough variance in the responses that it still makes sense to consider all three measures. It is likely that this is the result of allowing location MTTR to vary. However, if only the component maintenance requirements are allowed to vary then the relationships between flexibility measures is much weaker and is almost to the point of being totally unrelated. The one exception is the relationship between divisibility and volume flexibility which has a correlation coefficient of 0.72 indicating that it may be acceptable in this case to only consider one of the two measures. (figure 45)

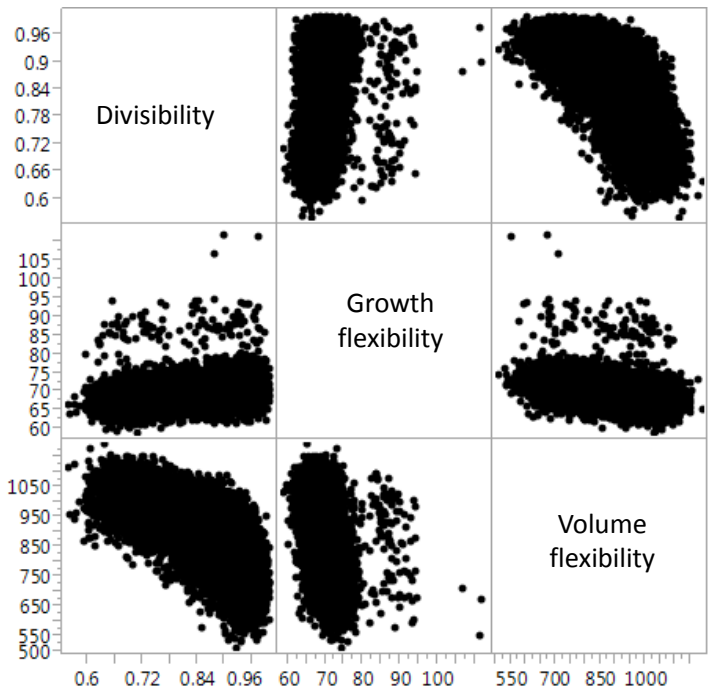


Figure 45: Matrix of plots comparing all flexibility measures when only maintenance requirements are varied

Table 36: Correlation matrix for flexibility measures when only maintenance requirements are varied

	Divisibility	Growth	Volume
Divisibility	1.0	0.2868	-0.7178
Growth	0.2868	1.0	-0.3812
Volume	-0.7178	-0.3812	1.0

4.3 Correlation between network properties

Using the 5000 random designs described in the previous section, the relationships between the different network properties can be explored. The primary goal here is to find a set of network properties that does not contain too much redundant information. The method for examining the relationships will be via correlation coefficients, therefore the goal is to find sets of responses with high pairwise correlation. To visualize this, scatterplots will be used to show the grouping of points. In most cases if the points group along a narrow 45 degree line, this is an indicator of high correlation. Additionally, the grouping will follow an exponential or logarithmic pattern in which case the correlation coefficient will be lower, but the relationship will still be considered significant.

From this a few clear relationships stand out. First the functional cyclically and graph energy of the P matrix show a strong linear correlation indicating that they are redundant measures. A correlation of 0.9825 indicating that only one of the two measures is necessary in most cases. (figure 46)

Additionally, max flow for the P matrix shows an exponential relationship with the previous two measures. This is an interesting result, and an argument could be made for calling maximum flow redundant as well. However, since the correlation is only 0.6204, maximum flow will remain. (figure 47)

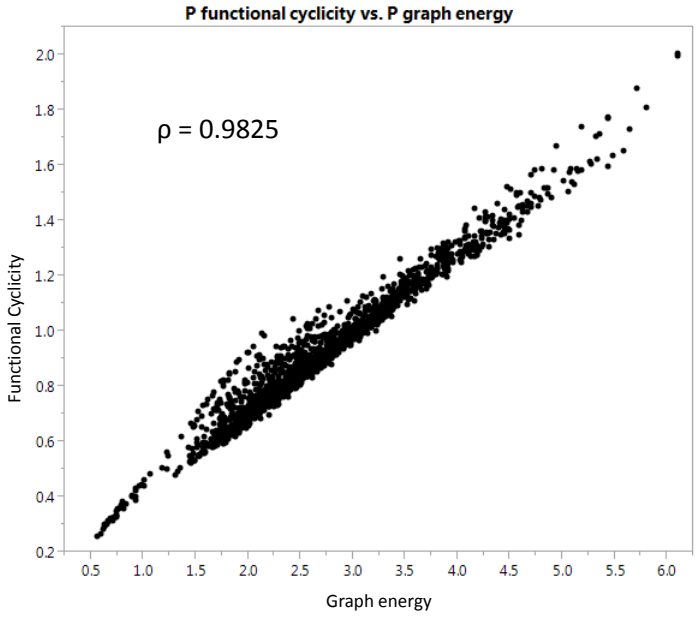


Figure 46: Relationship between functional cyclicity and graph energy

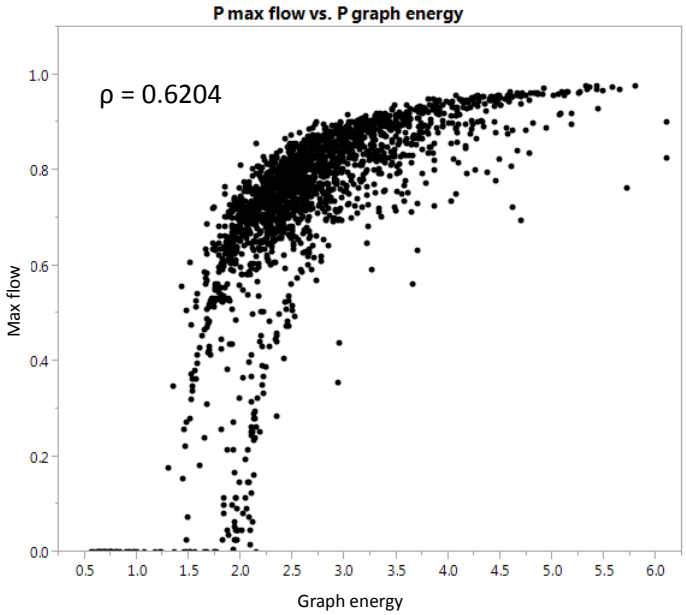


Figure 47: Relationship between max flow and graph energy

Additionally, many of the measures from the different N matrices are strongly related. In general all the responses from the N2 and N3 are highly correlated as are N1 and N4, meaning that only one model is necessary. In this case N1 and N2 are deemed sufficient. (figures 48 and 49)

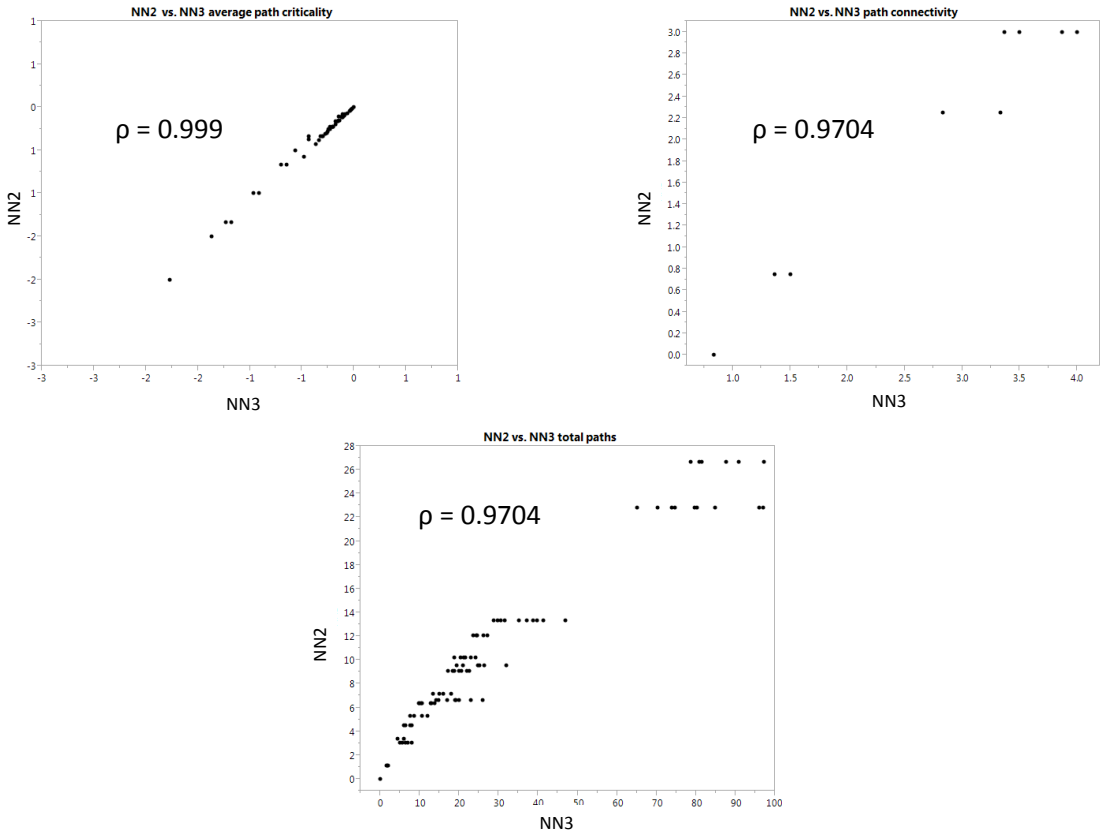


Figure 48: Three plots describing the relationship between responses for the N2 and N3 networks

It should be noted that even though the correlation between models of path connectivity is 0.9704 indicating that they are redundant, both responses are retained. This is due to the highly discrete nature of the responses with only four possible values from N2 and eight possible values from N3. While in general, the responses from N2 are used, in this case the response from N3 is also used because it contains more information. (figure 50)

Finally, it should be noted that the measures derived from the A matrix do not vary

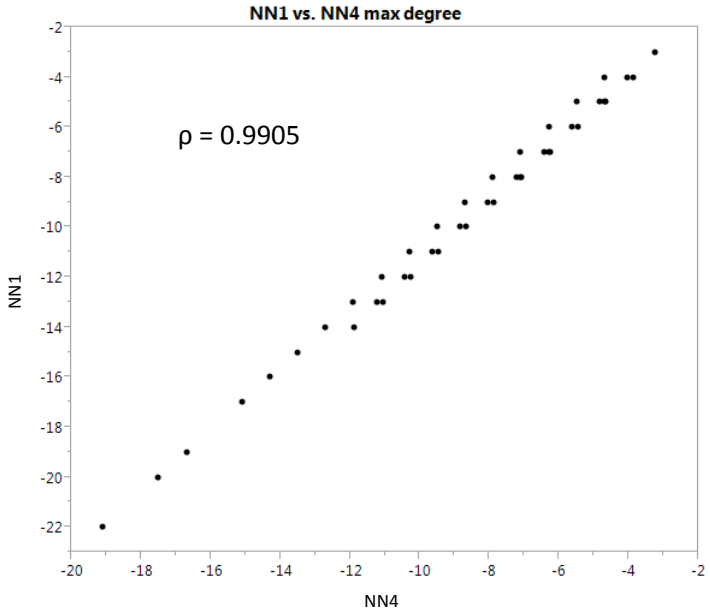


Figure 49: Relationship between maximum degree in the N1 and N4 networks

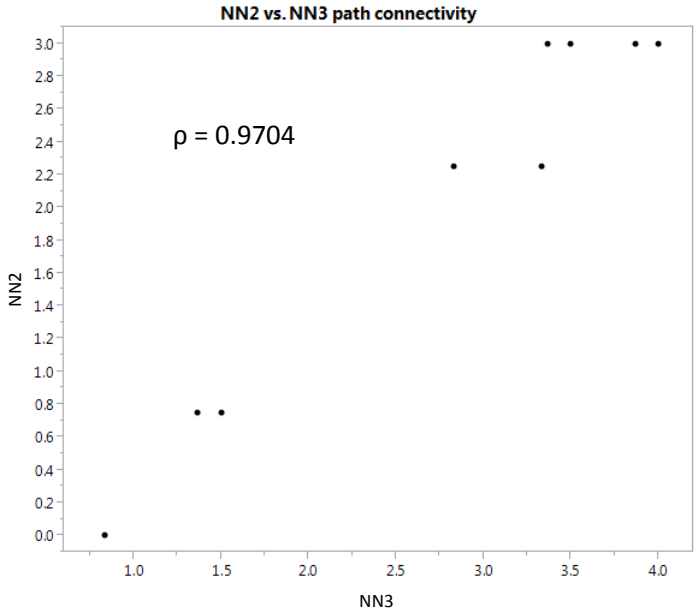


Figure 50: Relationship between path connectivity in the N2 and N3 networks

at all for this data set. One interpretation could be that this network does not provide anything useful. Alternatively, it can be due to the simplicity of the example problem. Either way, these measures will not be useful for this study and will be ignored. Through this process 14 measures are reduced to 8. They are as follows:

1. maximum flow through the P network
2. functional cyclically of the P network
3. algebraic connectivity of the P network
4. maximum degree in the N1 network
5. number of paths in the N2 network
6. most critical path in the N2 network
7. path connectivity of the N2 network
8. path connectivity of the N3 network

However, for the remaining network properties the relationships are too difficult to identify intuitively. In order to further reduce the dimensionality of the problem, the remaining network properties can be condensed using principal component analysis. In principal component analysis, the eigenvectors of the covariance matrix are used to obtain a set of uncorrelated, orthogonal, linear combinations of the responses. [89, 91](figure 51) In this way, the eigenvectors with the largest eigenvalues account for most of the variability in the response. In this case the first three components make up over 80% of the variability and can be used to represent the results of FLoRA NET to see multi-dimensional relationships when it is compared to the results of the simulations. For reference, the first three eigenvectors are listed below.

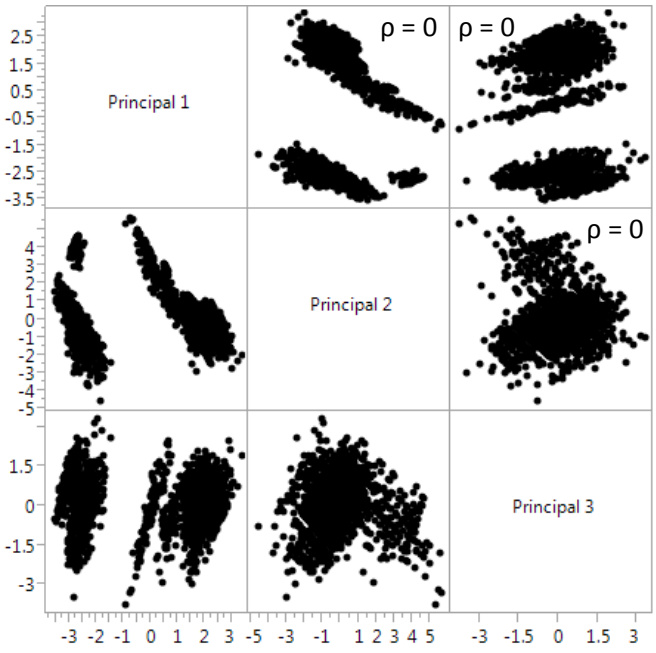


Figure 51: Matrix of scatter plots describing the relationship between the principal components

Table 37: First three eigenvectors of the covariance matrix of network properties

	Prin1	Prin2	Prin3
Pmaxflow	0.17834	-0.57873	0.35008
Pfunctional cyclicity	0.27065	-0.37097	0.49493
Palgebraic connectivity	-0.29079	0.41706	0.34513
NN1degree	0.09580	0.45293	0.69008
NN2paths	0.38927	0.04389	-0.04549
NN2critical path	0.41811	0.33861	-0.13469
NN2path connectivity	0.48869	0.12934	-0.07330
NN3path connectivity	0.48584	0.12273	-0.10762

4.4 Relationship between flexibility and performance

To reiterate, the first hypothesis was that measuring flexibility is a way to identify designs that perform well even in a disruptive environment. To test this hypothesis, the first experiment will use the previously described sample set of 5000 random designs to explore the relationship between flexibility, cost and performance. If the simulations were formulated properly then performance should increase with flexibility and cost should decrease. Additionally, these relationships should be more pronounced when there are disruptions than when there are none. This assumes that it is possible for inflexible designs to still perform well as long as there are no disruptions. Finally, it should be noted that correlation coefficients will once again be used to characterize the strength of the relationships and scatter plots will be used to visualize the results. It is expected that the hypothesis will only apply to some of the relationships between flexibility and performance, therefore for those relationships flexibility will be a good indicator of disrupted performance.

4.4.1 Flexibility vs. Cost

The first category of flexibility is divisibility, which is a measure of the systems resilience to depot failures. Primarily, divisibility is dependent on the distribution of maintenance responsibility over the sites. In general, if the distribution is fairly even, then most of the jobs will end up at sites with low MTTR as they will tend to have the shortest queues and lowest costs. Therefore, high divisibility should correspond to lower cost resulting in a negative correlation, which is the case in figure 52. However, when a depot fails, the more flexible systems will tend to continue operations and incurring costs while the less flexible designs will fail. The result will be that the systems with high divisibility will tend to have slightly higher costs when depots start failing. This will manifest as a weakening in the relationship between divisibility and cost as disruptions are added which is apparent from the reduction in correlation coefficient as the number of disruptions increases from zero to nine.

The second category of flexibility is growth flexibility which is modeled as the amount of time it takes the system to fully implement an upgrade. In general, the less time it takes to implement an upgrade the less it will cost, therefore high upgrade response time will correspond to high costs resulting in a positive correlation. Additionally, since the upgrade time is dependent on the MTTR of the maintenance sites the flexible systems will tend to have lower costs even when there are no disruptions which is apparent in figure 53. However, it can also be shown that the relationship between cost and growth flexibility is significantly stronger when there are disruptions. This is apparent from the increase in correlation coefficient from 0.58 to 0.72 when the number of disruptions goes from zero to three. It is also interesting that unlike divisibility the strength of the relationship does not drop off as much as more disruptions are considered. One possible explanation is that this is caused by the fact that growth flexibility is measured over a shorter time interval.

The third category of flexibility is volume flexibility, which is the systems maximum

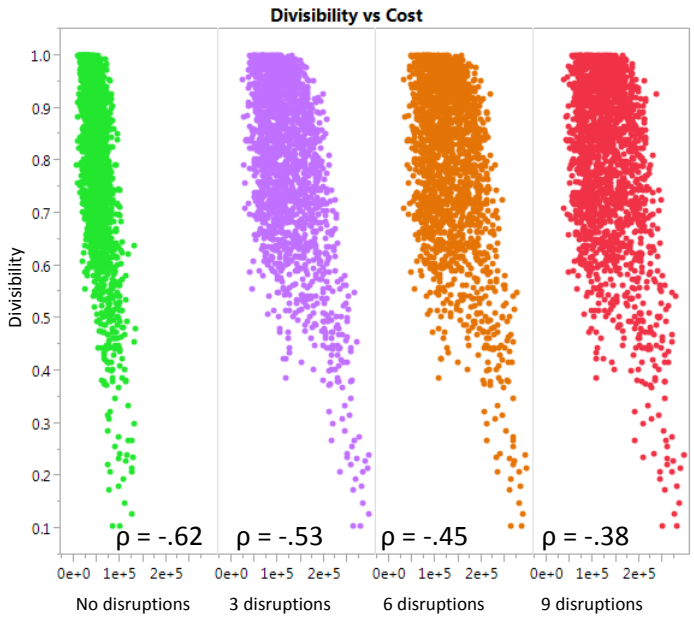


Figure 52: Relationship between divisibility and cost with increasing levels of disruptions

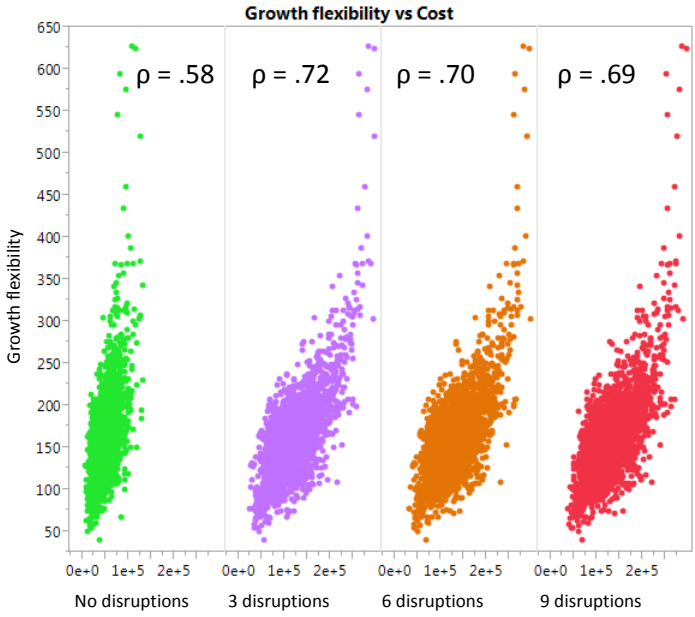


Figure 53: Relationship between growth flexibility and cost with increasing levels of disruptions

capacity as measured by the maximum operational frequency that the maintenance sites can support. In general, if the sites can process jobs quickly then they will be able to handle a larger increase in operational frequency. At the same time, the faster they process jobs the less they cost, therefore high volume flexibility should correspond to lower costs. This can be shown in figure 54 by the strong negative correlations. Additionally, in support of the hypothesis, this relationship strengthens when disruptions are considered as shown by the increase in correlation coefficient from 0.71 to 0.81. However, the relationship tends to weaken as more disruptions are considered. This can be explained by the fact that some disruptions, such as failed depots, will reduce the volume flexibility of the system which will result in systems that once had high flexibility also having higher costs than they did with fewer disruptions. It should also be noted that the relationship seems to be logarithmic indicating a diminishing returns effect where increases in volume flexibility have less pronounced advantages at some point.

4.4.2 Flexibility vs Availability

Since divisibility is measured as the number of depots that have failed once availability drops below 65%, high divisibility should naturally correspond to higher availability. This can be shown in figure 55 by the high correlation coefficients across the board. In fact for each level of disruptions the correlation between divisibility and availability is higher than between any other two responses. This would seem to make up for the fact that divisibility did not relate well to cost. Additionally, to further support the hypothesis, the relationship strengthens when disruptions are considered, and similarly to the previous cases the relationship tends to weaken as the number of disruptions are increased further.

Growth flexibility being defined by the time it takes for the availability to return to ideal after a component goes obsolete should correspond favorably to the overall

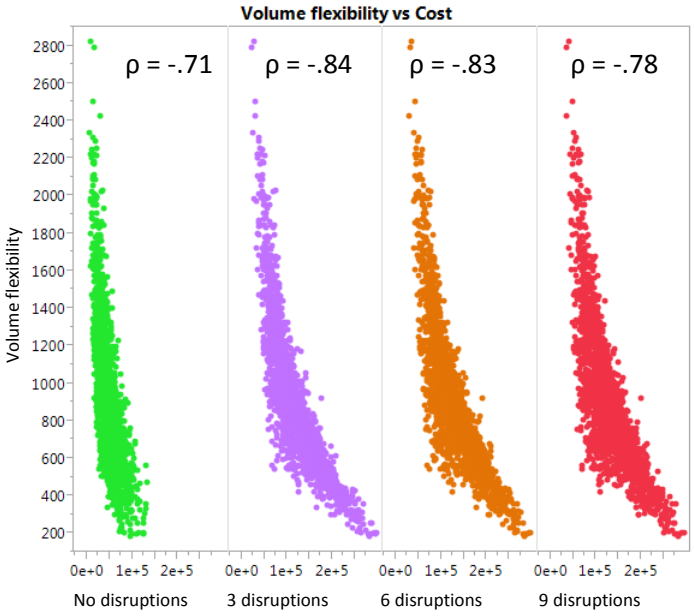


Figure 54: Relationship between volume flexibility and cost with increasing levels of disruptions

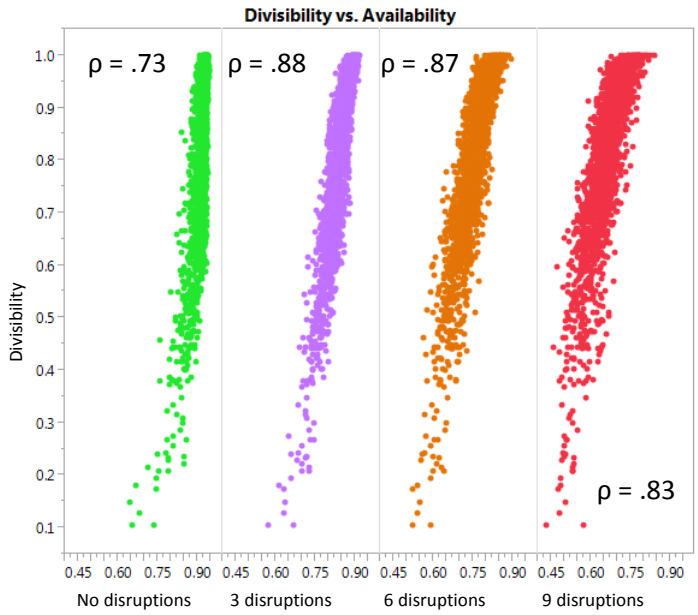


Figure 55: Relationship between divisibility and availability with increasing levels of disruptions

availability. This can be shown in figure 56 by the negative correlation relating fast upgrade response times with lower costs. However, the strength of the relationship is not as high as some of the other measures since growth flexibility is only measured on a short time frame while availability is measured as an aggregate over the entire life span of the system. As such, it is also not surprising that the relationship does not strengthen with the consideration of disruptions.

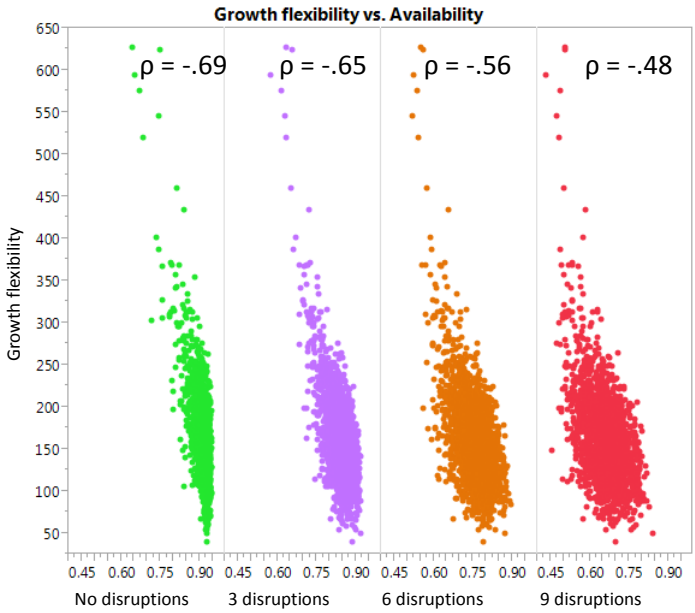


Figure 56: Relationship between growth flexibility and availability with increasing levels of disruptions

Finally, since volume flexibility is also related to a 65% threshold on performance, systems with high maximum capacity should also have higher average availability. While it can be shown in figure 57 that the correlation coefficients are positive, supporting this assumption, the relationship is not particularly strong. This can be explained by the diminishing returns effect due to the fact that availability cannot ever be above

100

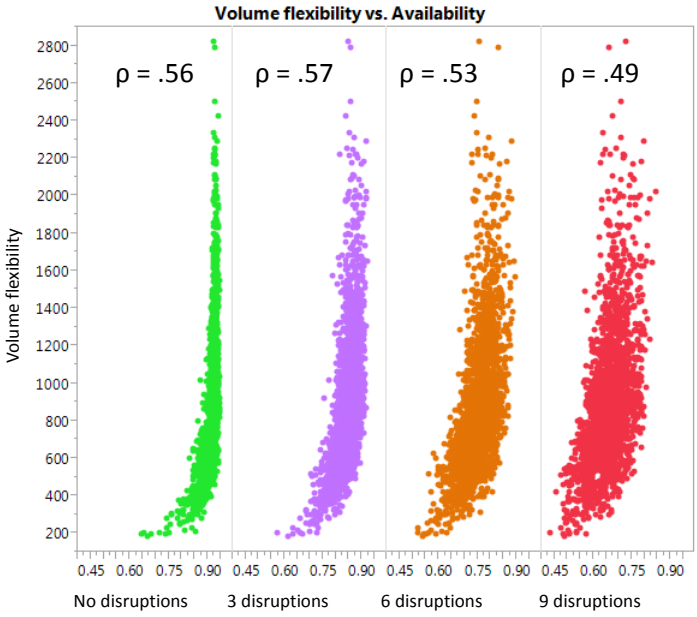


Figure 57: Relationship between volume flexibility and availability with increasing levels of disruptions

4.4.3 Summary of results

The scatter plot matrix in figure 58 summarizes the findings of this experiment. The result is that divisibility tends to correlate strongly with availability and growth and volume flexibility correlate strongly with cost. For these pairs the first hypothesis is supported as the relationships tend to strengthen when disruptions are considered. Additionally, in all cases the relationships tend to weaken when the number of disruptions is increased further. This can be explained by assuming that the more disruptions are considered, the more the behaviors of the different designs tend to converge. In this case all the designs will tend to fail by the time that four depots

have stopped working which corresponds to twelve random disruptions, therefore nine disruptions is the theoretical limit for this example problem.

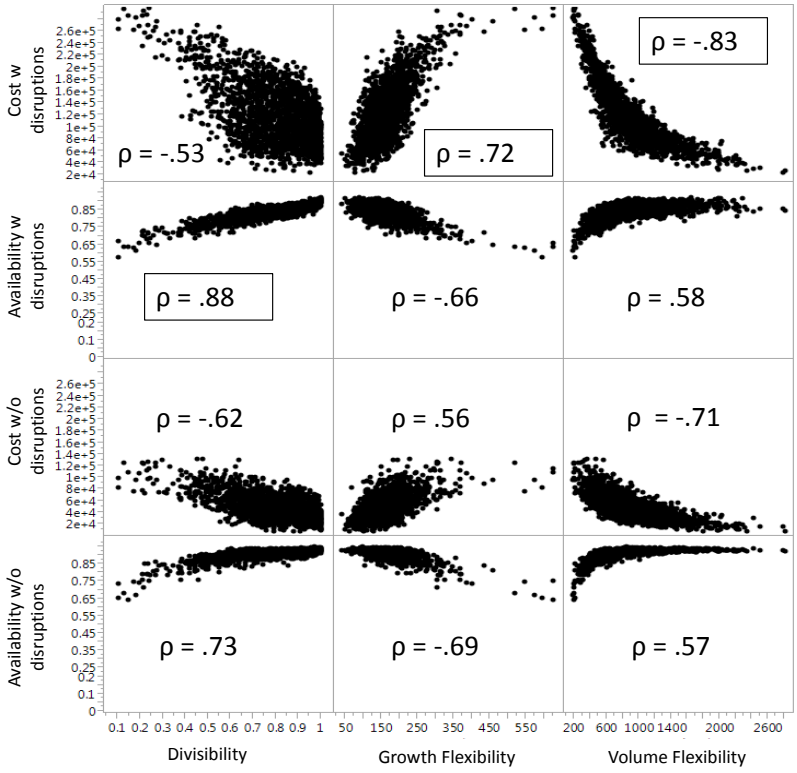


Figure 58: Matrix of plots showing cost and performance vs. flexibility with correlation coefficients

On final point is than in essence the correlation coefficient is a measure of the strength of a linear function fit between the two data sets. It is apparent from the scatter plots that there may be some non-linear relationships such as the logarithmic relationship between volume flexibility and cost. For these relationships the measure of strength is R^2 which is the square of the correlation coefficient describing the linear relationship between the actual data and the values predicted by a functional fit model. In general if $R^2 > \rho^2$ then the relationship between the two responses is better described by a non-linear function. Two such examples are as follows.

Three parameter exponential function

$$y = a + be^{cx} \tag{24}$$

Quadratic function

$$y = a + bx + cx^2 \tag{25}$$

Once again the best fits are obtained consistent with previous results. Cost is predicted by an exponential model of operational shifts with $R^2 = .83$ and availability is predicted by a quadratic model of the proportion of failed depots with $R^2 = .78$. The rest of the significant fits are summarized in the scatter plot matrix in figure 59

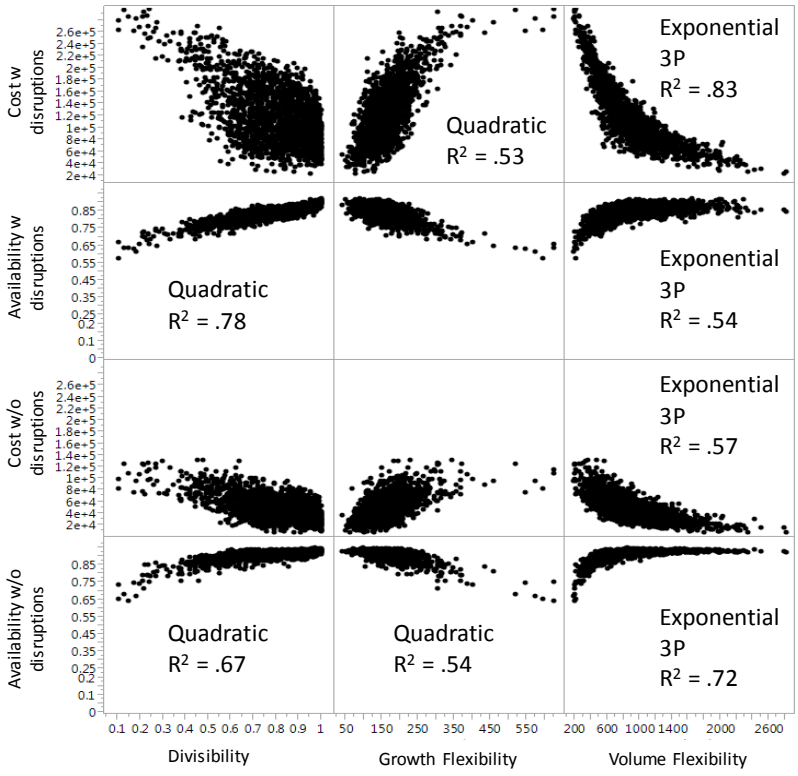


Figure 59: Matrix of plots showing cost and performance vs. flexibility with goodness of fit statistics

In conclusion, showing that relationships exist between flexibility and performance and that some of those relationships are more relevant when subjected to disruptions,

is a sufficient condition to substantiate the first hypothesis that flexibility can be used to identify good solutions.

4.5 Relationship between flexibility and network properties

The second hypothesis was that network properties could be used as heuristics for flexibility for the SoS maintenance planning problem. Similar to experiment one, this experiment will use the 5000 random designs to explore the relationship between flexibility and network properties by means of correlation coefficient. Unlike the relationships between flexibility, cost and performance, these relationships are expected to be weaker since, network properties are not physically analogous to flexibility which is the nature of heuristics. As such, meaningful statements explaining why the observed behavior is realistic will be hard to make. In summary, even though the correlation will be lower, if the coefficients are on the order of 0.4 to 0.6 they should still be usable for a design space down selection. However, if they are more on the order of 0.1 to 0.3 then a weak relationship does exist and a probabilistic approximation of the relationship might produce better results.

To reiterate, the eight network properties that were calculated for each of the 5000 randomly generated designs are as follows.

1. maximum flow through the P network (MF)
2. functional cyclically of the P network (FC)
3. algebraic connectivity of the P network (AC)
4. maximum degree in the N1 network (Deg)
5. number of paths in the N2 network (Npath)
6. most critical path in the N2 network (Cpath)
7. path connectivity of the N2 network (Pconn2)

8. path connectivity of the N3 network (Pconn3)

These will be compared to the following three simulated measures of flexibility.

1. proportion of failed depots
2. upgrade response time
3. maximum operational load

4.5.1 Correlations

There are twenty four possible relationships, and the results are summarized in the following table. Most of the relationships are very weak therefore, only the notable ones will be explored in more depth.

Table 38: Correlation between flexibility and network properties for the 5000 random designs

	MF	FC	AC	Pconn3
Divisibility	-0.0897	0.0268	0.0794	0.0703
Growth	-0.18	-0.2958	0.1495	-0.1124
Volume	0.2252	0.3294	-0.2388	0.1201
	Deg	Npath	Cpath	Pconn2
Divisibility	0.1347	0.0419	0.0829	0.0672
Growth	0.1133	-0.0723	-0.0574	-0.1052
Volume	-0.1912	0.0615	0.0235	0.1066

The first observation here is that none of the correlations are stronger than 0.4 and only the relationship between volume flexibility and functional cyclicity in the P network is stronger than 0.3. To examine that further figure 60 shows the relationships between the properties of the P network and volume flexibility. From this it can be shown that higher functional cyclicity tends to correspond to higher volume flexibility.

Similarly higher max flow tends to correspond to higher volume flexibility, which is the expected result as volume flexibility is supposed to be the system's maximum resource flow capacity. However, it seems that the values for max flow cluster around the value of 0.6 meaning that for high values of volume flexibility max flow is not a good indicator. Finally, while the results show a slight negative correlation between algebraic connectivity and volume flexibility, nothing is visually apparent from the scatter plot.

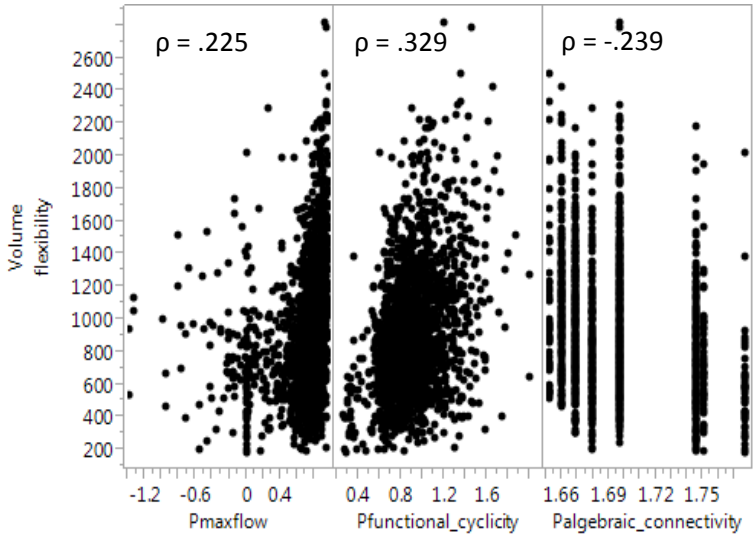


Figure 60: Scatter plots of evaluated designs comparing volume flexibility and the properties of the P network

While the results for the 5000 random designs are not particularly useful, it is possible that this is due to the nature of graph theory. It is reasonable to believe that the network properties will be more dependent on the placement of edges in the graph which are exclusively determined by the maintenance requirements of the components. As

such, varying site MTTR, component MTBF and operational frequency is likely to confound the relationship between network properties and flexibility. Therefore, it makes sense to explore the relationships in the context of only varying the maintenance strategies. Below are the correlation coefficients between network properties and flexibility for the 7776 designs from the second design space evaluation.

Table 39: Correlation between flexibility and network properties when only maintenance strategy is varied

	MF	FC	AC	Pconn3
Divisibility	-0.4	-0.2431	0.4838	0.4517
Growth	-0.1893	-0.2084	0.2767	0.1766
Volume	0.5171	0.5343	-0.6516	-0.4027
	Deg	Npath	Cpath	Pconn2
Divisibility	-0.0142	0.2055	0.0154	0.0149
Growth	-0.1229	-0.0829	-0.1607	-0.1796
Volume	0.3243	0.1553	0.3762	0.4046

It is immediately apparent that many of the correlations are much higher than before. For example the correlation between volume flexibility and algebraic connectivity is now 0.65 whereas it was previously 0.24. Algebraic connectivity is supposed to relate to the network's stability, therefore when faced with more components to service, one can assume that a more stable network will handle the change better. A similar argument can be made for why volume flexibility should relate well to algebraic connectivity and why functional cyclicity, which is supposed to relate to the network's ability to synchronize, should also correlate with the systems response to increases in operational frequency. Additionally, it was shown in section 4.1 that for this design space, divisibility and volume flexibility are negatively correlated, therefore it is expected that some of the same relationships that were found for volume flexibility

will also exist for divisibility. (figure 61)

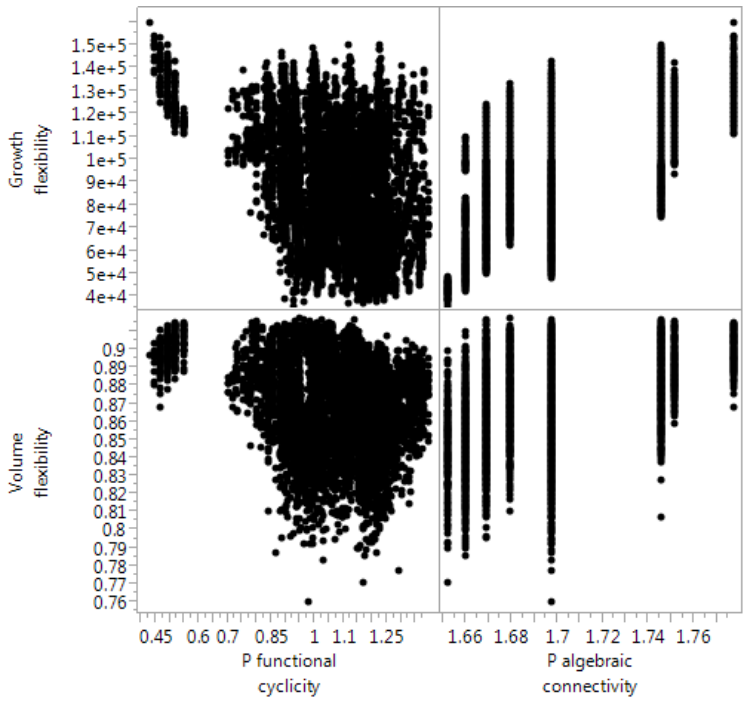


Figure 61: Scatter plots comparing volume and growth flexibility with the functional cyclicity and algebraic connectivity of the P network

Continuing on, it is no surprise that max flow still correlates well with volume flexibility. It is also not terribly surprising that divisibility and the maximum degree of nodes in the N network should be related. It was previously assumed that a node with high degree would be a possible choke point. It was then assumed that a network with choke points would more vulnerable to node failures, therefore it should be correlated with the system's response to depot failures. Finally, it should be noted that none of the network properties correlate particularly well with growth flexibility. This is not surprising as it was discussed in chapter 1 how growth flexibility is the hardest to generalize using network properties. (figure 62)

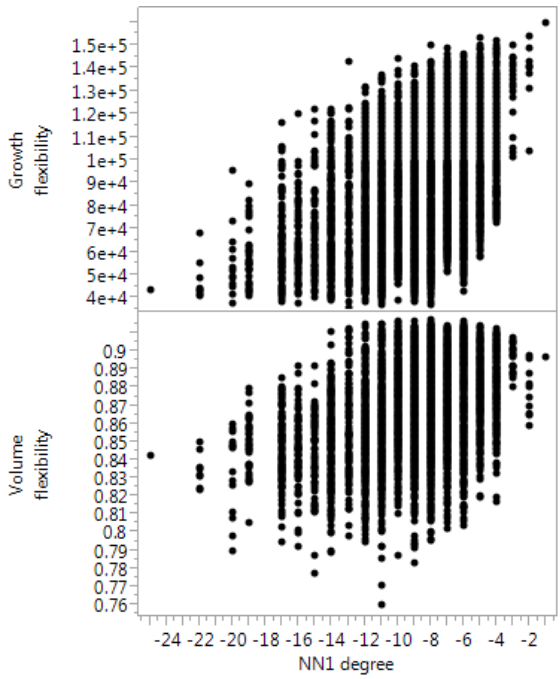


Figure 62: Scatter plots comparing volume and growth flexibility with the maximum nodal degree in the N1 network

To summarize, for the 5000 random designs the relationships are weaker than expected. However, for the design space where only the maintenance requirements are varied, the relationships are much stronger. The properties of the P network perform well, as does the maximum degree in the N network. The biggest problem is that nothing relates particularly well to growth flexibility. Additionally, even though the correlations indicate relationships of medium strength, it is hard to see more than an ellipse of dots in the scatter plots. Therefore additional methods will be explored.

4.5.2 Principal components

The next attempt was to look for higher order relationships between combinations of network properties. To do this, the three highest ranking principal components were compared to the flexibility measures.

Table 40: Correlation matrix relating principal components of FLoRA Net to flexibility measures

	Prin1	Prin2	Prin3
Divisibility	0.0607	0.1305	0.0695
Growth	-0.1227	0.205	-0.009
Volume	0.1231	-0.301	-0.0338

Examining the results shows that using principal components did not improve the correlation at all. Whereas the strongest relationships before were between volume and growth flexibility and functional cyclicity with a correlation of 0.33 and 0.29 respectively, now the strongest correlation is with principal number two with only values of 0.3 and 0.2 respectively. This would indicate that principal components are not a good reduction of dimensionality for this problem.

Alternatively, it is possible that the principal components will perform better with respect to the 7776 full factorial design space. For reference the eigenvectors of the three highest ranking principal components are defined below.

Table 41: Eigenvectors of the covariance matrix when only maintenance strategy is varied

	Prin1	Prin2	Prin3
P max flow	0.25402	-0.52533	0.45488
P functional cyclicity	0.4133	-0.22343	0.20946
P algebraic connectivity	-0.2946	0.44315	0.14298
NN1 degree	0.06025	0.48908	0.78431
NN2 paths	0.36193	0.09071	-0.09413
NN2 path criticality	0.35735	0.42664	-0.30756
NN2 path connectivity	0.45706	0.1597	-0.0533
NN3 path connectivity	0.45457	0.15091	-0.08561

Table 42: Correlation matrix relating principal components of FLoRA Net to flexibility measures when only maintenance strategy is varied

	Prin1	Prin2	Prin3
Divisibility	-0.112	0.5817	0.1409
Growth	-0.2068	0.1881	0.1217
Volume	0.4994	-0.4854	-0.2169

When examining the correlations, the strongest relationship for growth flexibility decreases from 0.65 to 0.49, indicating that algebraic connectivity would be a better heuristics. However, the strongest relationship for divisibility increases from 0.48 to 0.58 meaning that principal 2 is a better heuristic than algebraic connectivity was. The formula for principal 2 is as follows.

$-3.1758190659345 * Pmaxflow + -1.33059560885448 * Pfunctionalcyclicity + 14.0070443862538 * Palgebraicconnectivity + 0.161609310889171 * NN1degree + 0.0122158866658684 * NN2paths + 0.00885455968807403 * NN2pathcrit + 0.145289630836606 * NN2pathconn + 0.142895948865611 * NN3pathconn +$

(−18.6428016501704)

To summarize, in general the principal components correlate worse than the original network properties. The reason is that only a few of the network properties correlate strongly to begin with, therefore considering combinations of multiple responses will usually be worse than just considering the single best correlated one. However, in the case of divisibility where there are multiple network properties that correlate around 0.4 then principal 2 which is mostly an aggregate of the four strongest correlated responses will perform best.

4.5.3 Simulation risk

Finally, as was mentioned at the beginning of this section, if the relationships were not deemed strong enough then a probabilistic method should be used to approximate the joint probability distributions (JPDF) of the responses. After evaluating the network properties of a particular architecture it is then possible to use the JPDFs to calculate the probability that the flexibility will be high. To do this the MATLAB copulafit function was used [7] to fit JPDFs. A copula is a function that describes the relationship between the marginal distributions of two variables. However, these relationships cannot be used in the same way that a linear relationship would be. Being a probabilistic fit, knowing one variable will only result in a probability distribution on the value of the related variable. As such, the copulas cannot be used to predict the measures of flexibility. However, it may be possible to use the copulas to manage the risk involved for each design. Risk, in the simplest form is a trade off between the consequences of bad outcomes and the likelihood that they occur. [114] In this case the consequence is the expenditure of computational resources to evaluate a potentially bad design, and the likelihood is the probability that the flexibility is below a tolerable threshold given network properties. In this way the copula can be used to reduce the number of times the simulation must be used when conducting

the design space exploration, thereby saving time.

The process can be summarized as follows.

- For each flexibility measure fit a multi-variate copula to all the relevant network properties or principal components. Figure 63 shows what the joint probability distribution function (JPDF) contours looks like in two dimensions.
- For each flexibility measure determine a threshold that will serve as a cutoff for what is deemed acceptable. Since the simulation risk function transforms the response to a unit interval in order to sample the JPDF, the threshold can be generalized by a normalized value between 0 and 1.
- For each design, evaluate all the network properties. Then sample each JPDF for every value of the flexibility measure with the network properties fixed. The result is the marginal distributions of the flexibility measures.
- For each marginal distribution, generate the cumulative distribution function and determine the probability that the value will be above the previously stated threshold. The result is the probability that the current design will have an acceptable level of flexibility
- Determine what an acceptable level of risk is, and only run the simulation if the probability of acceptable flexibility is high enough.

The expected result should be that this method will filter out the majority of cases that exhibit less than optimal flexibility. Given the results from the previous chapter, this should in turn select a higher density of solutions with good performance. Granted it will also throw out many solutions that may be good. However, this is the cost of being more selective, and given that the goal is to pick a solution that is sufficient and can be adapted easily in the future, throwing out some good solutions is an acceptable price to pay for efficiency.

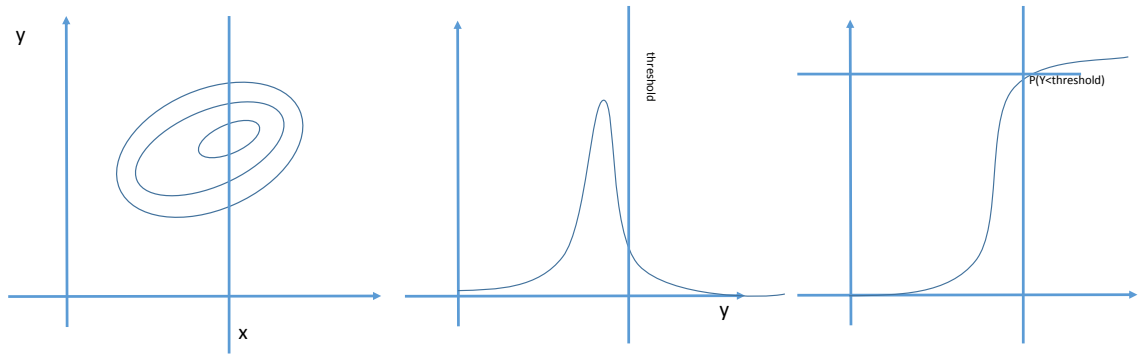


Figure 63: From left to right: A 2D joint probability distribution with a vertical line depicting a fixed x value. The marginal distribution on y with a line depicting a threshold for acceptable values. The CDF of y where the vertical line is the threshold on y and the horizontal line represents an acceptable level of risk

This was applied to the 5000 random designs and after some tuning of the thresholds, the following down selection was achieved. Figure 64 shows how the original design space exploration would have been conducted if simulation risk were used to filter designs. The parameters for accepting a solution were that it has a 40% chance of being in the top 10% of the solution space for each category of flexibility. These values were determined experimentally and in general must be found uniquely for each different problem. The result was an 8% increase in the average volume flexibility and a 7% increase in the growth flexibility. What is more striking is the 23% decrease in average cost and that only 2.8% of the design space was actually simulated.

To summarize, simulation risk is a promising method for selectively choosing which designs to expend additional computational resources on. It is not a perfect solution though, since it requires a data set that is a good representation of the solution space in order to fit distribution functions to. However, this is primarily a cost that is

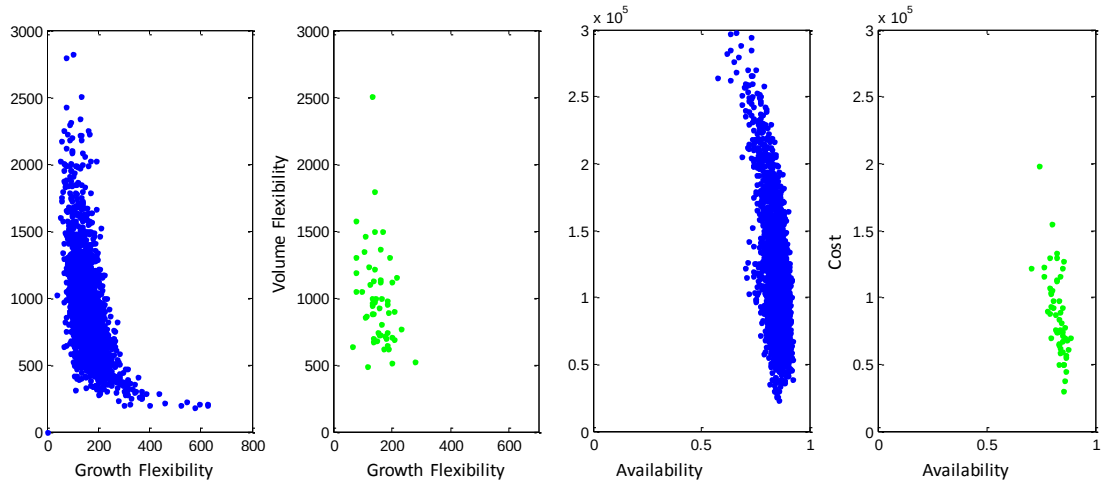


Figure 64: Blue: Response space for all solutions. Green: only the low risk solutions. Left: Comparison of flexibility response spaces. Right: Performance vs cost response spaces

incurred at the very beginning of the design process. Once it is done, all future analyses can be sped up ideally to the point that they can be done in a fraction of the time.

4.5.4 Summary of experiment 2

In conclusion, for the original set of 5000 random designs the network properties did not correlate sufficiently with the flexibility measures. However, when only the maintenance requirements of each component were varied the network model correlated much better with the flexibility measures. This is because the network properties are dependent on the topology of the graphs which is primarily driven by the maintenance requirements. It was determined that the task to task network was the best model of volume and divisibility, while no good heuristic for growth flexibility was found.

It had been discussed in section 4.2 that principal component analysis could be used to aggregate the network properties and reduce the dimensionality of the problem. However, this was shown to not be as effective in general. The one exception was with principal 2 of the 7776 design full factorial data set which correlated best with volume flexibility.

Finally, a probabilistic method was formulated for quantifying the risk of expending computational resources to simulate a poor design. While traditional goodness of fit metrics were not available for this method, it was possible to simulate a design space down selection on the 5000 random designs. It was found that while the method was time consuming to set up and required significant tuning of parameters, the results promised a potential 93% increase in efficiency.

4.6 Comparison of design space down selection methods

The final hypothesis was that an optimization algorithm could more efficiently explore the design space than the default method. This is more of conjecture than a formal hypothesis since only two methods will be compared which will likely be insufficient

to fully substantiate the above statement. In this experiment, the efficiency of the 5000 random design data set will be compared to a design space down selection using the NSGA II evolutionary algorithm. The two methods will be compared based on how many designs are evaluated in total and how well each method finds the best performing solutions. Finally, since this is a multi-criteria problem, the result of each method will be a set of non-dominated solutions that will serve as the design space down selection. It should be noted that since cost and performance are generally the primary metrics with flexibility being secondary, all the Pareto fronts will be evaluated from the perspective of cost and availability.

Nomenclature:

- P Performance / availability
- C - Cost
- F - vector of flexibility measures
- F1 Divisibility
- F2 Growth flexibility
- F3 Volume flexibility

Traditional LoRA optimizes cost and availability subject to constraints on a list of different metrics that are not being considered for this study. Therefore, in this design paradigm flexibility is not considered at all and for the purposes of this study it can be assumed that there are no constraints. The baseline optimization problem can be formally defined as follows.

$$\text{minimize} \quad C, -P$$

Figure 65 is a scatter plot of the results for NSGA2 running with a population size of 100 for 25 generations. There is a clear Pareto front of designs which seems to be rather angular (the highlighted points in the bottom left corner). This indicates that there is a small set of solutions that are likely to be optimal for the majority of scenarios. There is conceivably a scenario where near perfect performance is desired and cost is less of an issue, however the increase in cost is quite significant. Additionally, there is a scenario where less performance is needed and reduced cost is desired, however there are also steep diminishing returns in this scenario.

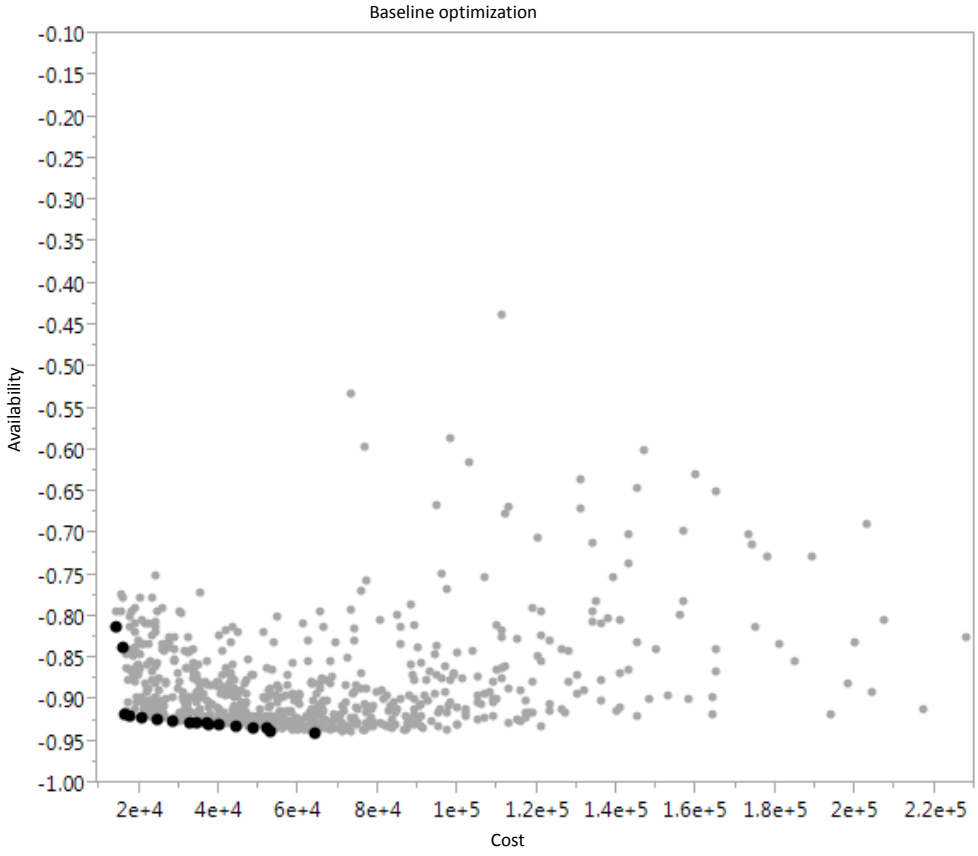


Figure 65: Results from the baseline optimization. Highlighted points represent dominant solutions.

When including flexibility into the formulation it makes sense to do so by considering it as an additional objective function. This is in contrast to setting flexibility as a constraint, since without a good sense as to how much flexibility is needed, the

resulting constraints would be arbitrary. The optimization problem for the multi-platform maintenance planning problem is defined as follows.

$$\text{minimize} \quad C, -P, -F1, F2, -F3$$

In this case F is a vector of three measures. Divisibility is the proportion of depots that can be disrupted before performance drops which should be maximized. Growth flexibility is the time it takes for performance to rebound after a component gets upgraded which should be minimized. Finally, volume flexibility is the maximum operational frequency that the system can operate effectively at, which should be maximized as well.

When comparing the Pareto fronts of the flexibility optimization to that of the baseline optimization, it appears that optimizing for flexibility is significantly worse (figure 67). While the optimal performance is nearly the same the optimal cost for a flexible system is 55% higher than the baseline optimum that does not consider cost. However, this can be explained since it is possible that those solutions have low flexibility and are therefore discarded by the flexibility optimization. This makes sense assuming that it is possible for some highly inflexible systems to still succeed given that only a finite number of disruption can be simulated. When the Pareto fronts for the flexibility measures are considered, it can be shown that the solutions that exist in the optimal region for flexibility also tend to exist in the region of low cost and high performance (highlighted points in figure 66).

When comparing the optimizations to the random sampling of designs performed earlier, the flexibility optimization finds a similar minimum cost but does so with 8% higher performance. What is most striking though, is that the flexibility optimization finds the optimal region in the 7th generation while the baseline optimization takes until the 9th generation, which compared to the random sampling of 5000 designs results in nearly an order of magnitude improvement.

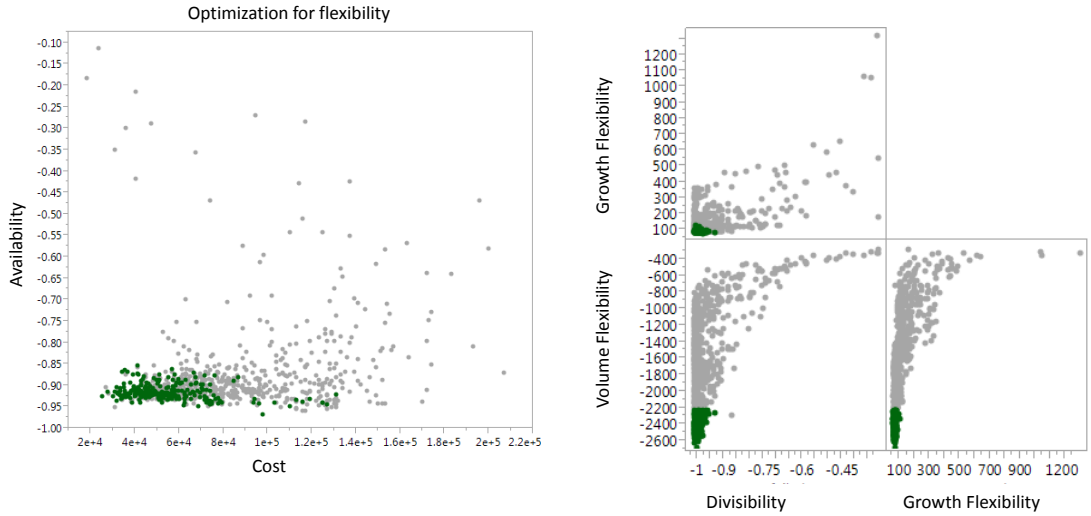


Figure 66: Results from optimizing with flexibility as an objective. Highlighted points represent dominant solutions in the cost vs availability solution space

The fourth down selection method is the one described in the previous section using a simulation risk filter on the 5000 random designs. As mentioned previously, the simulation risk filter down selected the design space to just 140 designs which is 80% less than the optimization for flexibility. However, the best performing solution remaining has 20% higher cost and 8% lower availability than the best performing solution from the flexibility based optimization. The question remains whether the reduction in performance is worth the increased evaluation efficiency. The cutoff point is likely to be different for each problem. In some cases where time is limited the increased efficiency will be valuable, while for other problems the optimizer will be efficient enough without sacrificing accuracy.

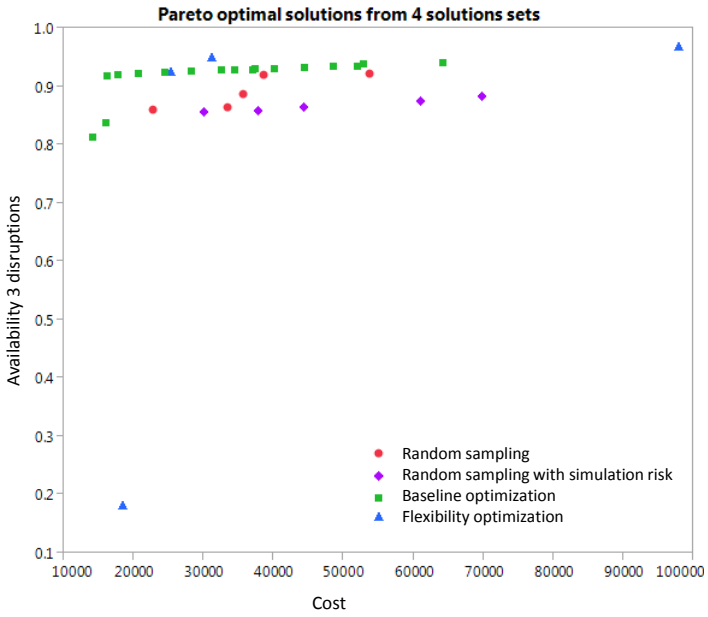


Figure 67: Comparing Pareto frontiers from three design space exploration methods

The final optimization paradigm uses the simulation risk formulation to heuristically select for designs with high flexibility. In this way the simulation risk acts as a

constraint on the optimization problem which can be defined as follows.

$$\begin{aligned} \text{minimize} \quad & C, -P, -F1, F2, -F3 \\ \text{subject to} \quad & risk < x \end{aligned}$$

The constraint is handled by a penalty function that when violated sets the cost arbitrarily high and the availability and flexibility to zero. If the constraints are violated then the simulation is not called at all thereby saving computational resources. The intended result is that the optimizer will initially throw away most of the bad solutions then find the regions where the low risk solutions exist and explore those in greater detail. However, figure 68 shows that the result is not as favorable as the algorithm fails to find the optimal region. What happens is that the algorithm starts with a random population of solutions of which only a couple are low risk. The next generation will start to focus on those favorable solutions and explore the region around them. After a few generations the algorithm has successfully found low risk solutions in the region of the original few. However, the population becomes saturated with these solutions and must simulate all of them thereby losing the initial computational advantage that was gained by using simulation risk. The problem is that with only a couple low risk solutions in the initial population there is no guarantee that they will be anywhere near the optimum. If they are not then the resulting Pareto front will not be anywhere near the true optimum. To mitigate this it might make sense to use a larger population size, however then the later generations that have found regions of low risk will take significantly longer to evaluate thereby negating any advantage gained by using an optimizer.

To summarize, the baseline version of LoRA uses an optimizer and does not consider flexibility, therefore as long as disruptions are likely to be a concern this method is not viable. However, when flexibility is added to the optimization the optimal cost is slightly higher, due to only considering flexible solutions, but the method is

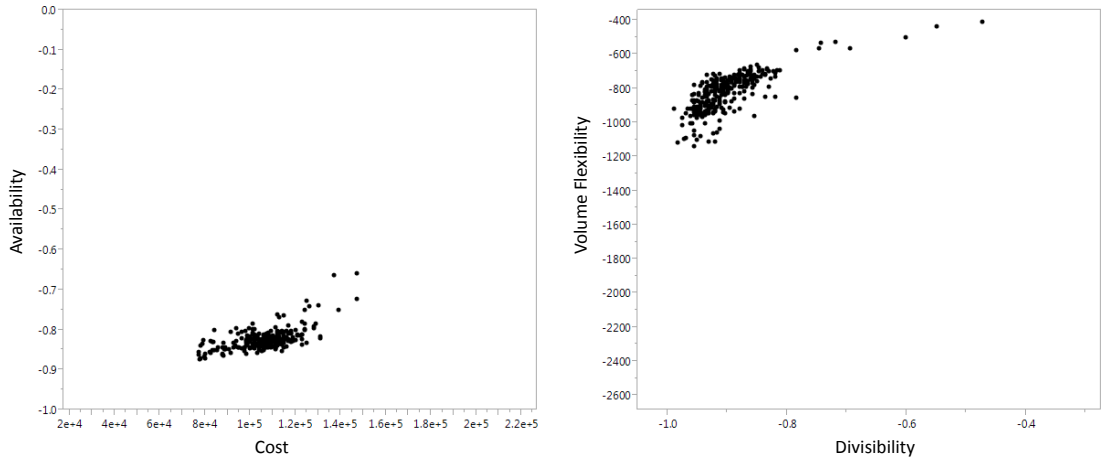


Figure 68: Results from optimizing with flexibility as an objective and simulation risk as a constraint

slightly more efficient than the baseline. When compared to the original method of randomly sampling designs, optimization is far more efficient and also produces better results. It was mentioned in the previous section, that the simulation risk method could also be used to rapidly down select the design space. However, when compared to the optimal solutions from the other methods, the results were significantly worse. Finally, it might make sense to combine the efficiency of simulation risk with the accuracy of optimization, but it was shown that this does not work well at all. In conclusion, the choice of down selection method is a trade off between accuracy and efficiency and will depend on the nature of the design problem and how much time there is to perform the evaluations.

CHAPTER V

APPLICATION OF THE METHODOLOGY TO A DESIGN CASE STUDY

This chapter will apply the proposed methodology to an example design problem for two purposes. First, to demonstrate how the methodology should be applied and second, to quantify the trade offs for using this method over current methods. To review, the steps of the methodology are as follows.

1. Problem definition
    - Definition of objective
    - Definition of baseline attributes
    - Determination of design variables
2. Evaluation of designs
    - Development of models
    - Definition of heuristics
3. Design space down selection
    - Evolutionary optimization
    - Application of heuristics
    - Determination of Pareto optimal designs

5.1 Problem definition

The first step of the methodology is to define the problem. For this case study, the problem is a multi-platform maintenance planning problem as described in chapter one. The need to consider disruptions to the system means that the new methodology should be used. Several types of disruptions that should be planned for were identified, including increases in operational frequency, component obsolescence, or maintenance depot failures.

5.1.1 Objective definition

The objectives can be defined as minimize support costs while maximizing platform availability and overall system flexibility. Flexibility is defined using the three part definition described in chapter one. The alternative designs are generated by varying the location where each component type gets repaired and replaced. This is consistent with a traditional level of repair analysis for fixed platform architectures, constant force structure and is constrained by having to use only existing maintenance sites.

5.1.2 Baseline definition

The baseline architecture is the one described at the beginning of chapter four. It has two operators with six platforms each, three depots, one intermediate, and one warehouse. Each platform has three out of the five different components.

Force structure	P1	P2	P3
O1	2	2	2
O2	2	1	3
W	2	2	2

Platform	Components
P1	C1 C2 C3
P2	C1 C4 C5
P3	C2 C4 C5

Component	MTBF
C1	500
C2	4000
C3	750
C4	1800
C5	2500

Location	Coordinates	Labor rate	MTTR	FH/Yr
O1	1500, 1500	50	15	420
O2	0, 2000	50	15	300
D1	500, 500	15	10	
D2	0, 1000	15	10	
D3	500, 1500	15	10	
I	1000, 1000	30	5	
W	0, 0			

5.1.3 Design variable determination

For each of the components the only design variable is the maintenance plan therefore there are $5^6 = 7776$ possible designs. This is a small number compared to real LoRA scenarios due to the limited number of components considered. The purpose of using such a small design space is so that multiple different methods can be implemented and compared in a realistic amount of time.

Variable name	Type	Range
C1 maintenance plan	Integer	[1,6]
C2 maintenance plan	Integer	[1,6]
C3 maintenance plan	Integer	[1,6]
C4 maintenance plan	Integer	[1,6]
C5 maintenance plan	Integer	[1,6]

5.2 Design evaluation

In this step of the methodology a preliminary screening of the design space should be done in order to determine how the flexibility measures relate to one another, and to define the relationship between the network model and the higher fidelity performance model. In general, this will involve sampling a small fraction of the design space at random. However, for this case study, the design space is small enough, therefore all 7776 possible designs could be evaluated using the FLoRA DES and FLoRA Net models. The advantage of using a small scale problem for the case study is that the best solutions can be known with absolute certainty, which aids in the assessment of the different down selection methods and the use of heuristics.

Many of the observations from this design space were mentioned in chapter four, and they will be summarized here.

- At the end of section 4.1 it was shown that for this formulation of the design problem growth flexibility is only weakly correlated however volume flexibility and divisibility have a stronger relationship. Therefore, it may be possible to only consider growth and volume flexibilities.

- In section 4.2 the set of network properties was reduced to eight loosely correlated responses. The same logic used for the extended design space applies to this case study, resulting in the same eight network properties. Additionally,

principal component analysis was used to reduce the dimensionality to three.

- In section 4.4 the relationships between network properties and flexibility measures was explored. For this design space, it was shown that the algebraic connectivity was best correlated with volume flexibility and the second principal component was correlated with divisibility. For these two measures of flexibility there were shown to be reasonable network based heuristics, but no such measure was found for growth flexibility. In this case, either simulation risk should be used or divisibility could be simulated directly. However, it was shown in the previous chapter that simulation risk is not compatible with evolutionary optimization. For this case study, optimization will be used, therefore simulation risk cannot be used. Either way, only simulating one out of three measures of flexibility is likely to save a significant amount of computational resources.

5.3 Design space down selection

In the final step of the methodology the Pareto front is set aside and one or two candidate solutions are chosen in order to investigate further and eventual implementation. In general, for multi-criteria problem such as this one, a weighting scenario must be defined that describes the customer's priorities with respect to which attributes are most important. However, scenario definition tends to be a very subjective step in the methodology. Therefore for this example problem it will be sufficient to say that since cost and availability are the primary metrics only designs that are on the cost vs availability front should be kept. However, cost and performance evaluations are dependent on the actual distribution of disruptions to the system, which is very hard to predict accurately. Therefore, when flexibility is measured, it will be assumed that cost and performance values are not available and the decision will be made based on flexibility alone. After that an estimation of cost and performance can be used as a

tiebreaker between similarly flexible designs.

(It should be noted that all optimizations are run using the NSGA II default settings with a population size of 50 for a maximum of 15 generations.)

5.3.1 Full factorial design space

The first result to look at is the set of all 7776 possible designs and see where the actual best solutions are. Figure 69 is a scatter plot of cost vs. availability for the entire design space. The highlighted points are those that lay on the Pareto front and the circled points are those that lay in the most favorable region of the design space characterized by the "elbow" in the curve. Of those one is circled that represents a very good solution that also has fairly high flexibility as can be seen in figure 70. It is interesting to note that none of the highlighted designs have the highest possible volume flexibility though most of them are in the next tier down. To comment on the computational cost of the full factorial design space evaluation requires 777600 runs of FLoRA DES to evaluate disrupted cost and availability and an additional 5598720 runs to evaluate flexibility. The total of over 6 million runs is estimated to have taken 34 days to evaluate.

5.3.2 Baseline optimization

The second down selection method assumes that only cost and availability are relevant and does not optimize for flexibility. This is consistent with traditional LoRA methods that don't consider disruptions at all. The result is a very quickly converging optimization that succeeds in finding most of the actual Pareto front not considering flexibility. (figure 71) Once again a couple solutions are circled to represent the likely best designs. It should be noted that the circled solution from the full design space is not found by this optimizer. The two circled designs in this scenario have slightly better divisibility but worse growth and volume flexibility. However, in terms of computational efficiency, it finds most of the Pareto front in only 133 unique design

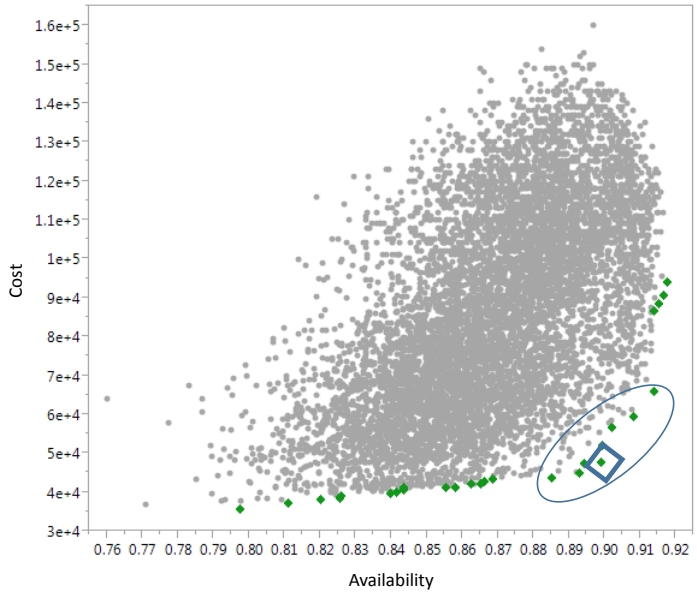


Figure 69: The complete design space with non-dominated points highlighted

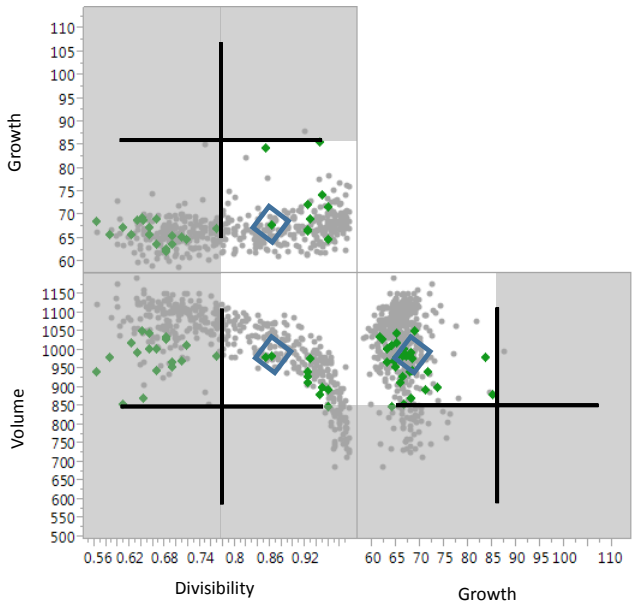


Figure 70: Non-dominated points with respect to all five metrics. Highlighted points are non-dominated with respect to cost and availability

evaluations which requires 13300 evaluations of FLoRA DES taking an estimated 2 hours to complete. Figure 72 is a plot of the number of unique designs that the optimizer evaluates per generation of the algorithm. It is apparent from there that the algorithm has mostly converged by around the seventh generation.

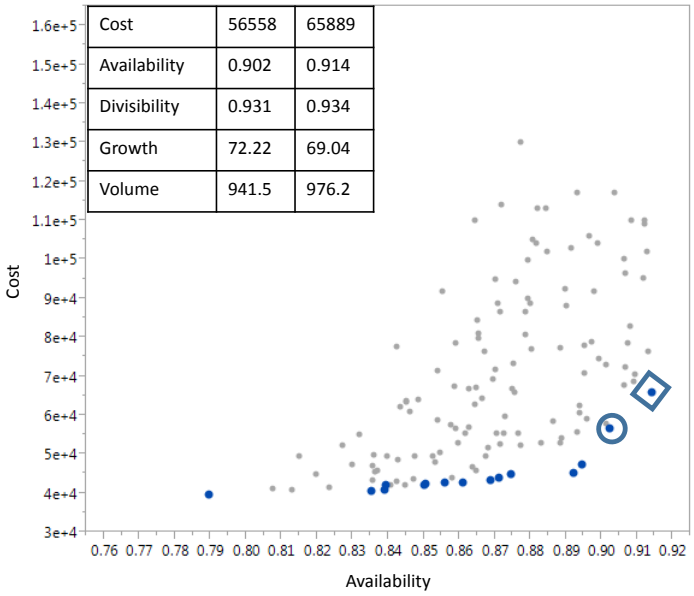


Figure 71: Designs evaluated by the baseline optimization with non-dominated points highlighted

5.3.3 Optimization with flexibility

The third down selection technique considers the three measures of flexibility as the primary objective functions. The expected result is that the algorithm will find solutions that have optimal flexibility, but when cost and availability are estimated they will be slightly worse than the baseline designs. Figure 73 is a scatter plot matrix of the three flexibility measures compared to one another. The solutions marked as

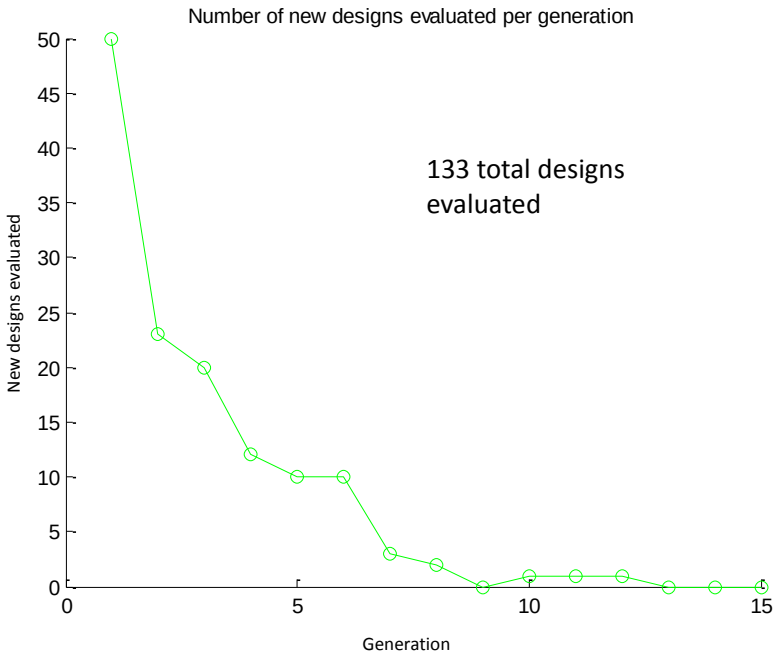


Figure 72: Convergence of the baseline optimization

triangles are all non-dominated with respect to flexibility, and the highlighted ones are selected assuming an even preference between the three measures of flexibility. Then disrupted cost and availability is estimated for those nine designs and the two remaining solutions are circled in figure 74. The results compared to the baseline optimization are summarized in the table below. The design in the diamond has better divisibility and growth flexibility but at the cost of slightly worse volume flexibility and higher cost.

Table 43: Difference between designs found by optimizing for flexibility and the baseline optimization

	Design 1	Design 2
Cost	+9.20%	+24.60%
Availability	0%	0.88%
Divisibility	+2.25%	+1.82%
Growth	-4.48%	+3.81%
Volume	-0.66%	-0.25%

When considering the convergence rate of this method figure 75 shows that 135 solutions were evaluated which requires 97200 runs of FLoRA DES. Of those, 9 were also evaluated for cost and availability, requiring 900 additional runs. In total, the 98100 runs are estimated to have taken about 13.5 hours which is nearly 7 times as long as the baseline optimization, but it is still nearly 60 times more efficient than the full factorial evaluation.

5.3.4 Flexibility based optimization with heuristics

The final down selection was done using the same objective functions as the previous method however, instead of evaluating divisibility and volume flexibility directly the previously defined heuristics were used evaluated with FLoRA Net. For volume flexibility, the algebraic connectivity was used and for divisibility, the second principle

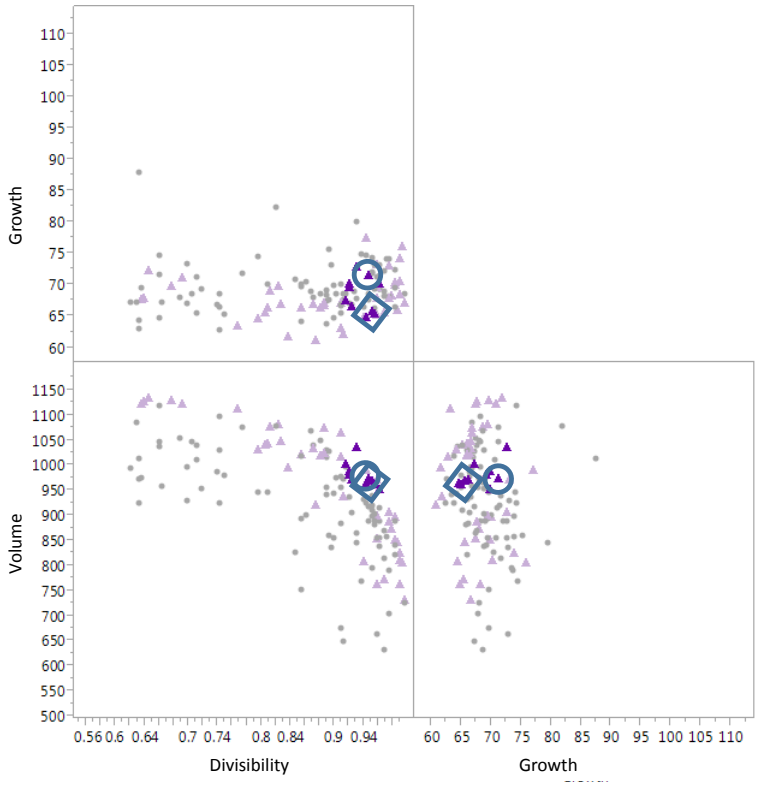


Figure 73: Flexibility scatter plots of designs evaluated by the flexibility based optimization. Non-dominated points are in purple. Down selected points are highlighted.

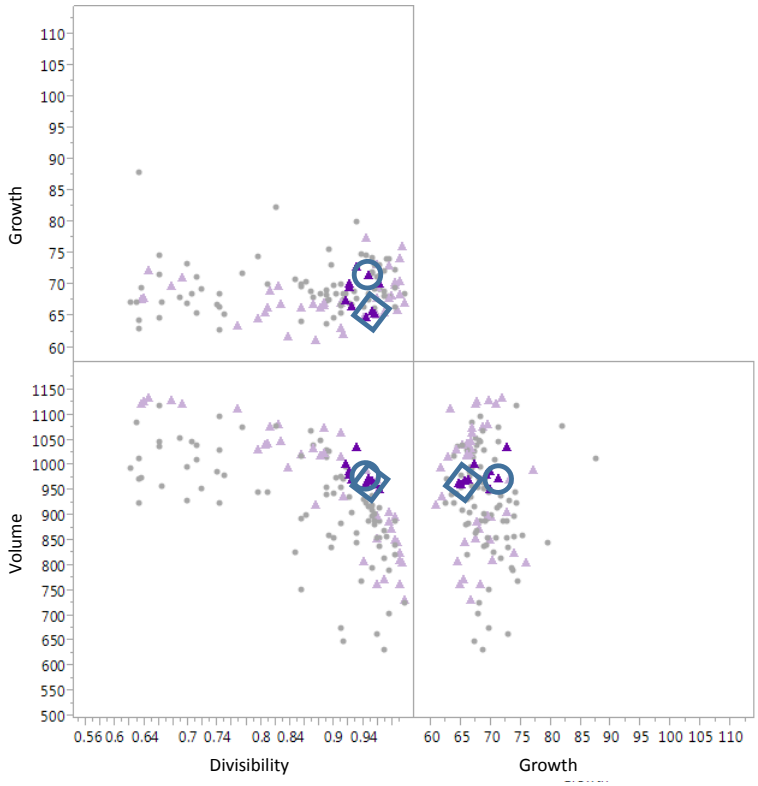


Figure 74: Cost and availability of designs evaluated by the flexibility based optimization. Chosen designs are circled

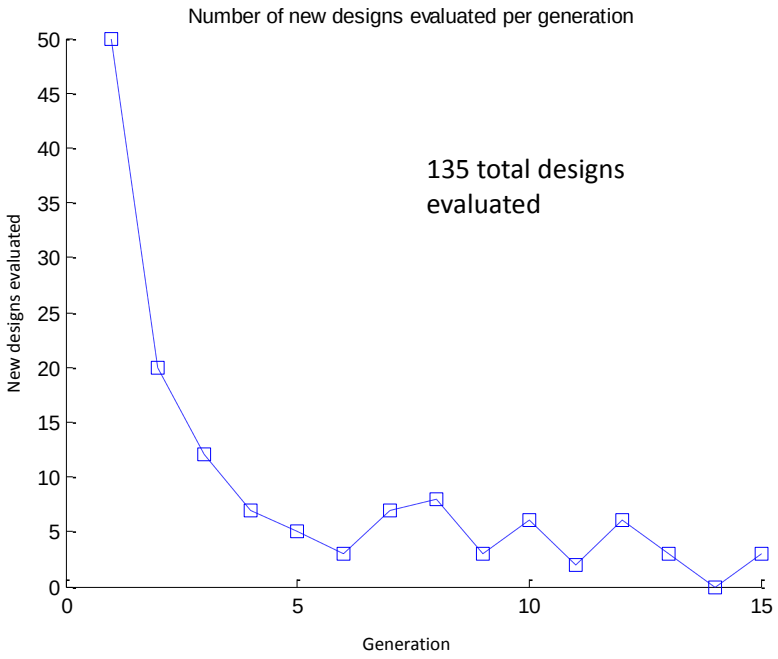


Figure 75: Convergence of the flexibility based optimization

component of the entire response set was used. It should be noted that since the correlation between algebraic connectivity and volume flexibility was negative, the heuristic must be minimized instead. The immediate result is that each design evaluation would require 231 fewer runs of FLoRA DES. The expected tradeoff is that the use of heuristics will prevent the optimizer from finding the same best solutions as before and the resulting chosen designs will be slightly worse.

Figure 76 shows all the designs evaluated by the optimizer in terms of growth flexibility and the two heuristic measures. Non-dominated designs are shown as red squares, and the highlighted ones are the down selected set of solutions. This is done prioritizing growth flexibility since it is not a heuristic, and giving equal preference after that to the volume and divisibility heuristics. The remaining flexibility measures and estimates for cost and availability are then evaluated for these five solutions, and while there are two that stand out in terms of flexibility (figure 77), only one has reasonable cost and availability (figure 78). When compared to the flexibility base optimization, it preforms slightly worse overall, though it actually manages to get better growth flexibility.

Table 44: Difference between designs found by optimizing with heuristics and optimizing for flexibility

Cost	2.10%
Availability	0.33%
Divisibility	-1.47%
Growth	-4.00%
Volume	-4.98%

What is more striking is that this optimization evaluated fewer unique designs than the previous method sand still found comparable solutions. 117 unique designs were evaluated requiring only 53703 runs of FLoRA DES and 117 runs of FLoRA Net which are negligible in comparison (figure 79). 860 additional runs were then required to

further evaluate the down selected designs, resulting in a total of 54563 runs. This is estimated to have taken about 7.5 hours to complete which is nearly half the time of the previous method. However, if a heuristic for growth flexibility could have been found then the computational savings would have been several orders of magnitude worth of time.

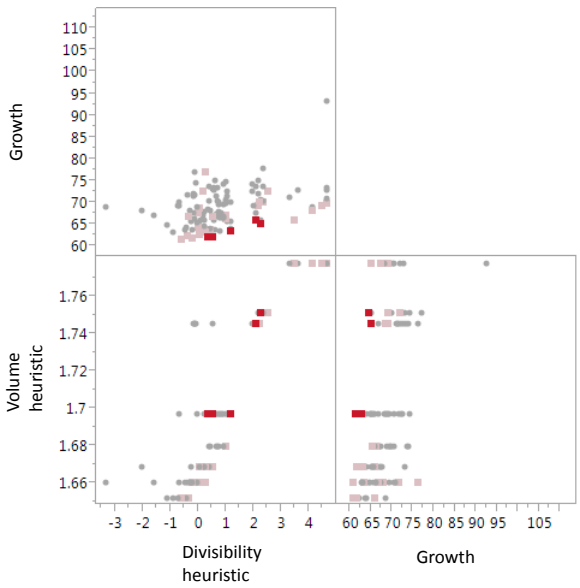


Figure 76: Heuristic measure scatter plots of designs evaluated by the flexibility based optimization with heuristics. Non-dominated points are in red. Down selected points are highlighted.

5.3.5 Summary of observations

Starting with the caveats, first, it is very hard to compare the above methods due to the fact that the actual chosen design will be heavily dependent on the priorities of the customers. While each method chose a slightly different set of optimal solutions, there was some overlap, and for a given scenario any one of those solutions could be

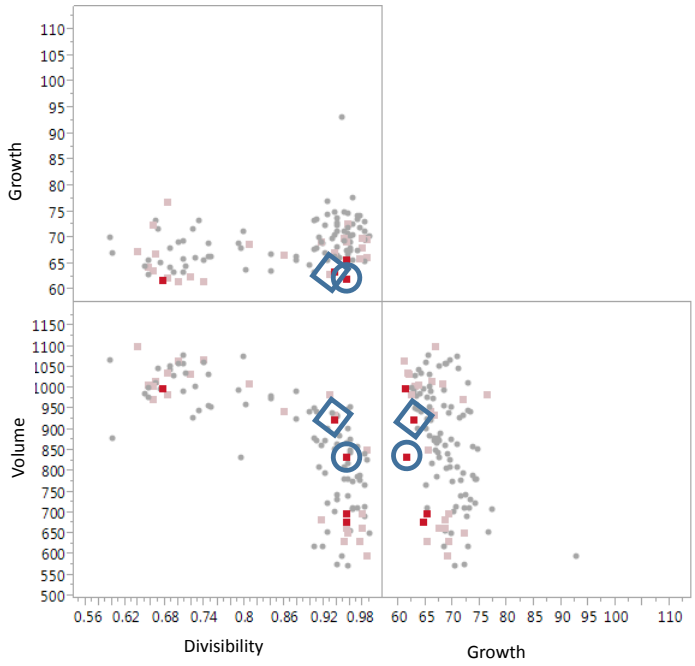


Figure 77: Flexibility scatter plots of designs evaluated by the flexibility based optimization with heuristics

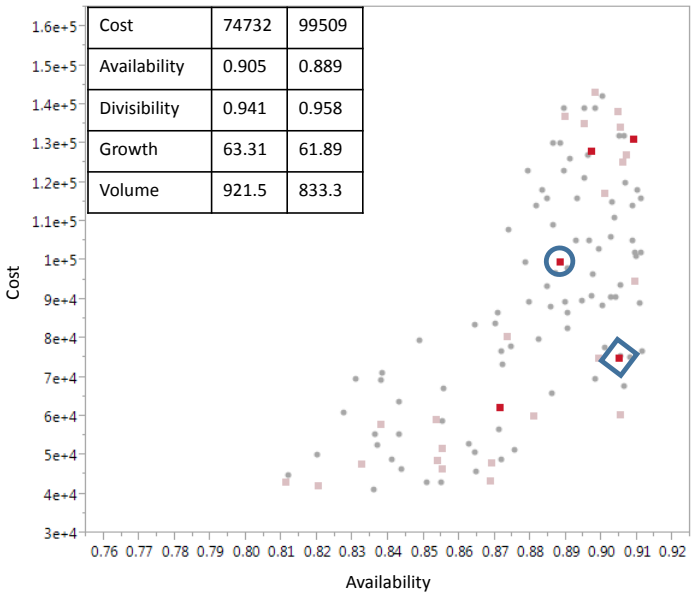


Figure 78: Cost and availability of designs evaluated by the flexibility based optimization with heuristics. Chosen designs are circled.

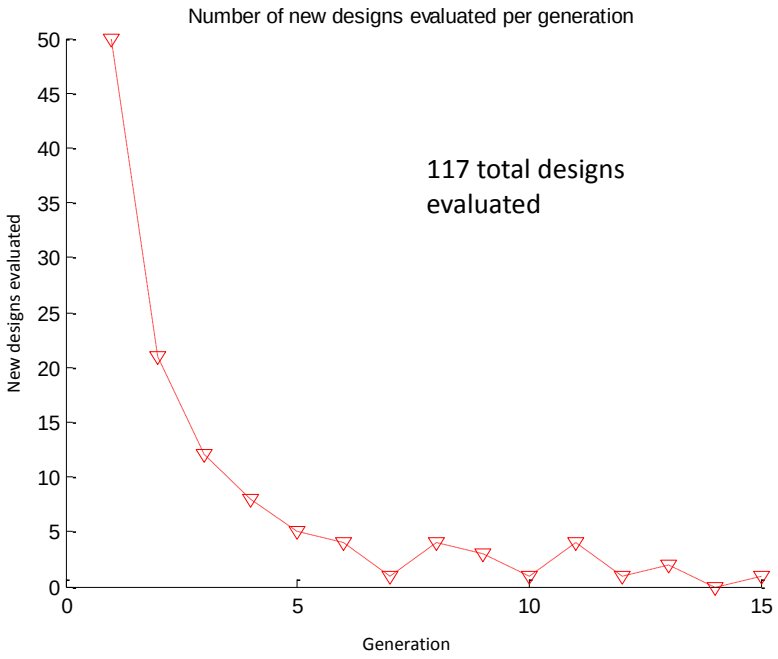


Figure 79: Convergence of the flexibility based optimization with heuristics

ideal. Figure 80 and figure 81 shows how the chosen solutions relate to one another in terms of all five metrics. The second issue with the comparisons, is that all of this is predicated on the assumption that the cost and availability measures are meaningful which are based on an assumed set of disruptions. However, it is generally not possible to accurately predict the set of disruptions. In this case one would only be able to evaluate flexibility and have to make decisions based on that alone.

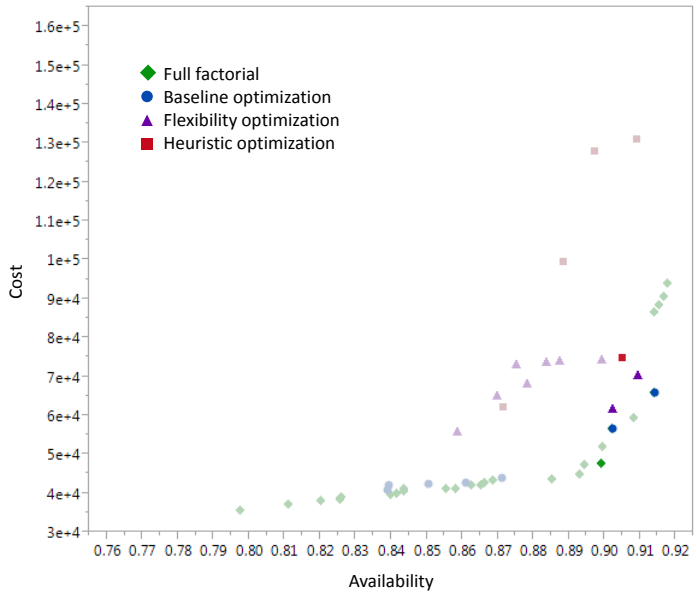


Figure 80: Cost and availabiltiy of designs chosen by each down selection method

If there are no performance and cost evaluations then there is no means to compare the flexibility based method with the baseline. That leaves the computational cost as the most realistic means of comparison. Figure 82 plots the convergence rates of each of the three optimizers against one another and it appears that all three optimizers converge around the same time which is generation six or seven. Therefore, the biggest difference is then in terms of the number of simulation runs required by each

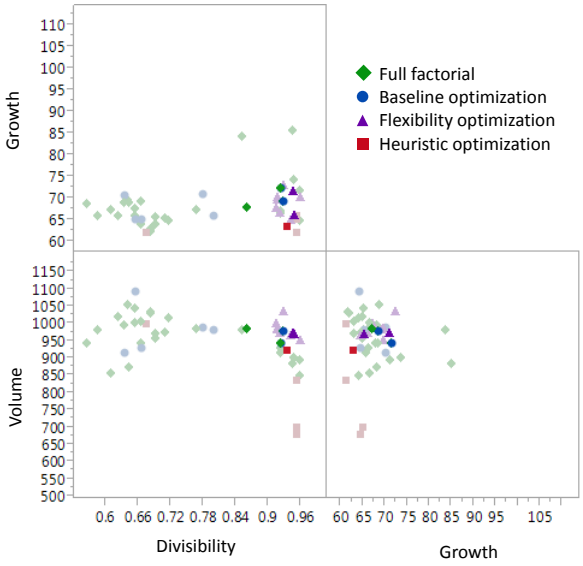


Figure 81: Flexibility of designs chosen by each down selection method

method. It was previously mentioned how the baseline optimization required an order of magnitude fewer simulation runs than the flexibility based methods. However, about half of that disadvantage was recouped by using heuristics. In either case using an optimizer was at least two orders of magnitude more efficient than a full design space evaluation. In general though, the design space will likely be too big to evaluate and an optimizer will be the only option. In conclusion, as long as evaluating performance in a disruptive environment is not a viable option, a flexibility based assessment should work, and if network heuristics are available then they can provide significant computational discounts.

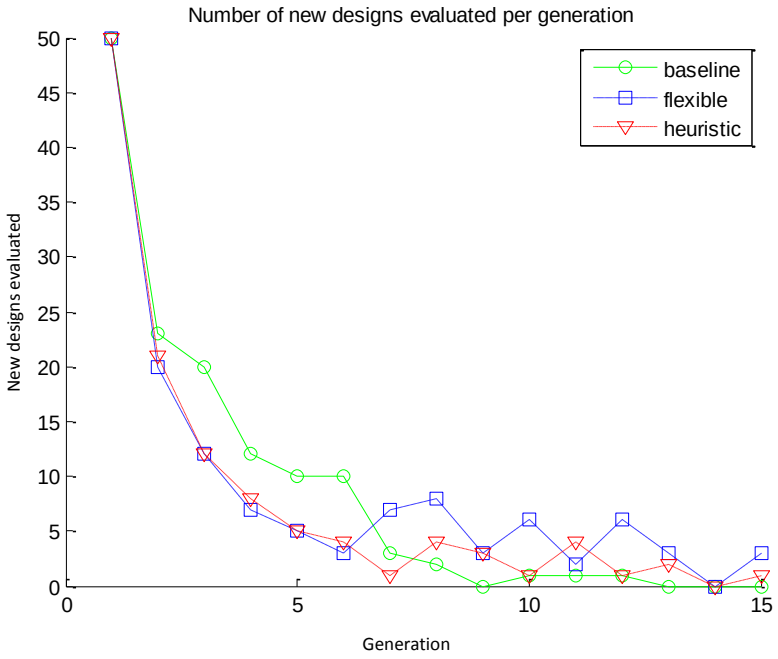


Figure 82: Comparison of convergence rates for each of the optimization methods

CHAPTER VI

CONCLUSIONS

6.1 Summary of findings

To summarize, the overarching purpose of this research was to develop a method for performing maintenance planning for multiple platforms types that share a set of common components. The primary objective is to determine where each component will get serviced in the most cost effective manner. However, a problem was identified that such an integrated logistics system would be more susceptible to disruptions such as component obsolescence. The increase in problem complexity made it so that traditional level of repair analysis methods would no longer apply. By drawing a comparison between the multi-platform maintenance planning problem and systems of systems, a new area of design methodologies became available for use. Hence, the research objective became to improve the existing SoS design methodologies and adapt them to the current design problem.

The following were three research questions that were posed representing gaps in the literature for solving the problem.

1. How can the system be evaluated in the context of a disruptive environment?

   To answer this question, it was hypothesized that a method based predominantly on prioritizing flexibility would result in designs that perform better in the presence of disruptions. It was then determined from the literature that a single definition of flexibility would not work. Flexibility is a multi-faceted system attribute and is actually an umbrella term that represents all properties that relate to how well the system performs with respect to disruptions. As such, three generic categories of flexibility were identified for the maintenance

planning problem and a simulation was developed to measure it.

In order to test this hypothesis, an experiment was conducted to determine how flexibility relates to the performance and cost of the system. First, it was found that for some sets of designs the flexibility measures are somewhat correlated, but that they all generally correspond to better performance and lower cost. However, when the three flexibility measures are mostly independent, there are real tradeoffs between them. Finally, it was shown that while flexibility can correlate with the cost and performance of the system when there are no disruptions, the correlation becomes even stronger when disruptions are considered. Therefore, when the set of future disruptions is unpredictable, flexibility is a better measure of the system's response to the average set of disruptions than its undisrupted performance.

2. Given that SoS modeling is expensive and the design spaces are large, are there ways of reducing the cost?

   Evaluating a small subset of a relatively small version of the maintenance planning problem, took nearly a month to complete. This issue of excessive simulation costs is a very real issue for large scale complex systems. Therefore, the second contribution of this thesis was to develop computational heuristics for SoS evaluation that are orders of magnitude faster to evaluate than the available simulations, but at the cost of reduced accuracy. It was hypothesized that for the multi-platform maintenance planning problem, a network model would provide a method for generating heuristics. It is likely that such a method would apply to any system dependent on resource or information flow between independent entities. A network model was developed that can be used to evaluate 14 different network properties 10^5 times faster than the discrete event simulation.

To test this hypothesis the network model was compared to the discrete event simulation to explore the relationships between the two. It was determined that strong relationships do not always exist, however when the design variables primarily affect the network topology, then the relationships are stronger. However, it is entirely possible that given a better network model, the relationships could be even stronger. In this case the network properties can be directly used to explore the design space instead of the computationally expensive simulation. The best solutions are then simulated in order to get more accurate estimates of flexibility. However, when the correlation is not strong enough, a method for using the network properties to determine the probability that the flexibility is sufficiently high and only simulate designs with high probability was shown to have some merit.

3. Given that it may not be possible to improve the modeling costs significantly due to the complexity of systems of systems, are there efficient methods of down selecting the number of designs so that the computational costs are concentrated only on evaluating the most promising designs?

   This question was in reality part two of the previous question because the literature on computational problem solving frequently talks about heuristics hand in hand with optimization algorithms. Therefore, it was suggested that optimization would be the appropriate method for down selecting the design space as an additional way to reduce the computational cost of SoS design space exploration. More specifically, it was proposed that an evolutionary algorithm would be the appropriate optimization method for the multi-platform maintenance problem. It is also likely that evolutionary algorithms are the best available method for SoS problems in general.

   Three different versions of optimizations were applied to an example design

problem. The first was an optimization of disrupted cost and availability even though it is assumed that such evaluations are not particularly accurate. As such this method was considered one baseline method in addition to fully evaluating all possible designs. The second method was to optimize for flexibility and the third was to optimize for heuristic values of flexibility. The end result was that all the methods were able to find a similar set of optimal designs, which is the best possible result. As expected the flexibility based method required the most simulation runs because it needs to evaluate the most metrics. Finally, the network heuristics were shown to be effective substitutes when optimizing for flexibility, and even though heuristics were only available for two of the three flexibility measures it was still twice as efficient.

6.2 Summary of contributions

The main contribution of this thesis was the development of a methodology for planning maintenance for multiple platforms with common components subject to disruptive conditions. This methodology is based on SoS design philosophies that focus on the need to rapidly implement a design that can be easily changed over time. In order to do this, a framework for measuring flexibility was proposed and applied to the maintenance planning problem. By designing flexibility into the system, the goal is to maximize the time between design updates. At which point, the methodology is intended to be repeated and the system is reevaluated in light of the current environment.

The secondary contribution of this thesis is in the area of modeling and simulation. Generally speaking, simulations designed to evaluate large complex systems require an immense amount of computational resources. This was found to be the case with the discrete event simulation that was developed to evaluate the service costs and platform availability of a maintenance system. Therefore, it was proposed that a

network model would be more efficient, at the cost of greater abstraction and reduced fidelity. Additionally, it was shown that an evolutionary algorithm would be a suitable method for exploring the design space and reducing the number of potential designs to a manageable number.

In conclusion, the multi-platform maintenance planning problem considering disruptions is not a well studied problem and as such there really are not any good methods for solving it. Therefore, trying to compare this methodology to an existing one would not be particularly meaningful as this one was designed to answer questions that are not being answered right now. This is also partly due to the fact that there isn't a single consensus SoS design methodology as sufficient data on real systems of systems is not available yet. However, this thesis proposes a very simple methodology that attempts to address some of the philosophical issues with current methods.

On a smaller scale, simulation and design space exploration are generically used for most design problems, and this thesis attempts to make improvements on those methods. Specifically, using a network model as a heuristic for a more complicated simulation was shown to work very well, when the modeling is done properly. It was shown to represent a potential five order of magnitude increase in efficiency. To put it in context, this would reduce an evaluation process that would have originally taken one year to only taking five minutes.

6.3 Future work

There is definitely room to expand on the work started in this thesis. The author would like to suggest a couple of areas where future work could be done.

- Explore more types of systems

  For this study, the effort involved in implementing the method for a single system was large enough. However, demonstrating this method for other systems

would do a great deal to solidify the claims made in this document. For example, military systems were mentioned a number of times in this document and would be a good place to start. Alternatively any SoS with a strong dependence on information or resource flow between elements would be a good candidate for this method.

- Develop a method for SoS network description

  The networks represented here are but a sampling of the many overlapping networks that can describe an SoS. Evaluating more of them would likely provide a more accurate picture and result in network measures that better correlate with the simulated flexibility measures. This thesis, as well as several other studies that were surveyed in this document, make the assumption that a network description of the system can be produced such that measurements can be made. Each of these studies chooses a method for defining a network but does not make it the focus of the research. It must be assumed that the measurement is only valid if the network description is valid. Therefore, it would be useful to make a study of the alternative methods for network definition, taking the measurement methods as constant, in order to develop a better tuned methodology.

- Further exploration of the iterative nature of the methodology

  There are two points in the methodology where iterations are called for and this document does not do a good job of exploring the effects of such a process. First, the iterative down selection process could be useful when multiple levels of model fidelity are available, and even the lowest fidelity models don't do a sufficient job reducing the number of alternatives. However, for this study only one simulation was available and the design space was small enough that it could be fully explored even at the highest available fidelity. Second, is how the

methodology is repeated over the course of the system's life span. It would be interesting to know how subsequent iterations of the methodology relate to one another, and how the system iteratively improves over time. For this study it was assumed that subsequent iterations can be considered independent and the issue was not explored.

APPENDIX A

OVERVIEW OF GRAPH THEORY

A basic overview of graph theory will be useful for understanding how networks are represented and evaluated mathematically. [78]

A.1 Graph

A graph is mathematical representation of a set of objects and their interconnections. The objects are called the "nodes" and the links are called "edges". If the edges are given a direction then the graph is called a "digraph". If the edges are given values representing something like capacity or transition time, then the graph is called a "weighted graph" Figure 83 shows the same basic graph as a digraph and again as a weighted graph. In this way it is possible to model the entities and connections in the network as well as the direction and properties of the resource flow or relationship.

A.2 Adjacency matrix

It is very hard for a computer to interpret pictures of graphs therefore, it is necessary to compress the information into a readable format. The easiest method is to generate a vector of all the edges in the graph described by the starting and ending nodes. As it turns out this is the sparse form of an NxN matrix for a graph with N nodes, where the rows and columns represent the nodes and the entries represent the edges . The entries in the matrix are Aij = 1 if there is an edge from node i to j and zero otherwise. If the edges lack direction then Aij = Aji and the matrix is symmetrical. If the edges have weights then the 1's are replaced by each edge's weight. Figure 84 takes the three graphs from the previous example and generates adjacency matrices for them.

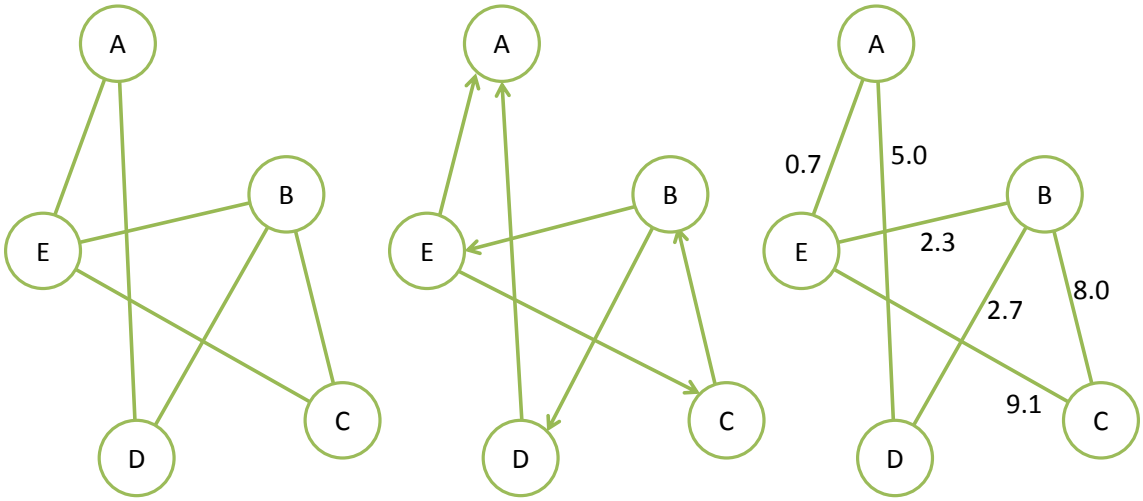
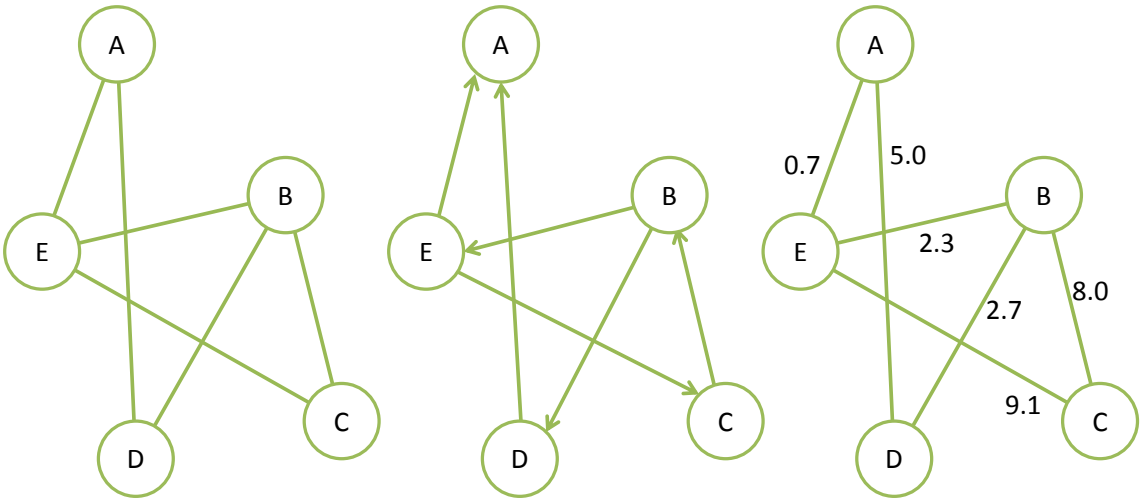


Figure 83: Example graphs: graph, digraph, weighted graph



$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 5.0 & 0.7 \\ 0 & 0 & 8.0 & 2.7 & 2.3 \\ 0 & 8.0 & 0 & 0 & 9.1 \\ 5.0 & 2.7 & 0 & 0 & 0 \\ 0.7 & 2.3 & 9.1 & 0 & 0 \end{bmatrix}$$

Figure 84: Example adjacency matrices: graph, digraph, weighted graph

A.3 Degree

For each node, the degree is the number of edges incident to it. The indegree is the number entering the node and the outdegree is the number leaving the node. Nodes with high degree tend to be more critical to the network than those with low degree. For example, in figure 85 the green node has a degree of three and is very critical to the network. Alternatively the red node has a degree of one. It can be assumed then that the single edge is also critical to the network. The distribution of degree across all nodes can be an indicator of how connected the graph is, and how easily it will be to disrupt the network. For example the average degree is very easily evaluated for a graph with m edges and n nodes $\frac{m}{2n}$.

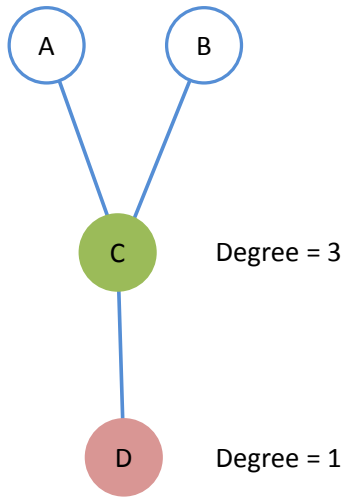


Figure 85: Example graph with average degree 1

A.4 Path length

To get from one node to another, the path consists of the edges that are traversed. The smallest number of those edges that is needed for that path is called the shortest path length. The characteristic path length (CPL) is the average over all shortest paths between any two points and is a good indicator for the time or cost it would take to get from one point to another. The betweenness centrality of a point is the number of shortest paths that go through that point which is an indicator of its centrality, importance, or criticality to the network. [73]

As mentioned in the previous chapter the paths through a network are of relevance in the measurement of flexibility. Paths in a network can represent the ability to move resources from one place to another, an unbroken line of communication, or a successful chain of sequential tasks, the presence of which generally indicates that the system is working as expected.

Cycles are a subset of paths that begin and end at the same node. They are of relevance to network theory as they can represent and unbroken feedback loop, or an iterative process in the network. The drawback for considering cycles is that the algorithms designed for finding them tend to be less efficient.

A.5 Cliques

A n-clique is any n nodes that are all mutually connected, meaning that each node is connected to each other node in the clique by an edge. The number of n-cliques (usually n = 3 is used) is a good indicator of the connectedness of the graph. The clustering coefficient of a node is a measure of how close its neighbors are to being a 3-clique. The average clustering over all nodes is an indicator of how well distributed the network is. It can be assumed that well distributed networks are more resilient to disruptions as the probability that a critical element will fail is less. The global clustering coefficient is the ratio of completed triples in the graph to the number of

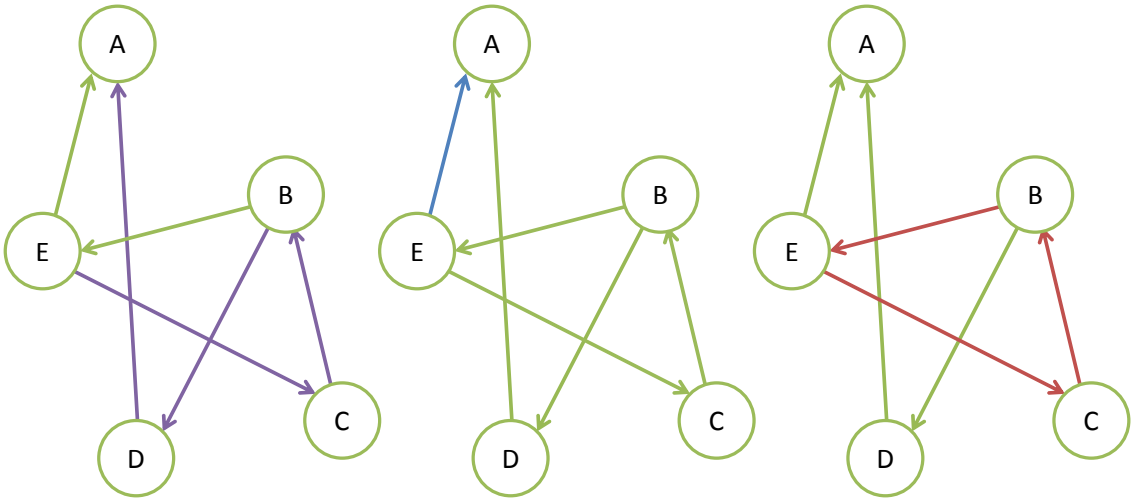


Figure 86: Examples of paths and cycles: (from left to right) A path from E to A, The shortest path from E to A, nodes B,C, and E are a cycle

open triples (a set of 3 connected nodes that only has two edges). From figure 83 nodes B, C, and E form a closed 3-clique, and the global clustering coefficient is 0.5.

A.6 Connectivity

As mentioned previously, the connectedness of a network can be used as an indicator of how easily it will be to interrupt its performance. The property of connectivity is a direct measure of the redundancy of elements in the network. Edge connectivity is the minimum number of edges that must be removed to break the graph into at least two disconnected subgraphs. The same can be said for node connectivity. For digraphs there is a similar concept called strong connectivity that considers if there is still a path connecting all nodes. However, if there is no viable path but the graph is still connected if no directions are considered, then the graph is considered weakly connected. Connectivity is generally an indication of how resistant the network is to disruption for networks that must remain connected to function. In figure 84 the basic graph has connectivity of 2 as either the edges incident to nodes A, D or C are removed, or nodes B and C are removed. The digraph however, can be weakly disconnected by removing nodes B, C, or E, or by removing any of the single inbound edges to nodes B, C, D or E.

A.7 Network flow

Weighted digraphs can be used to model the flow of resources through a network from one point to another. The max flow through the network can be used as an indicator of the networks maximum throughput or capacity. As mentioned previously, the higher the capacity of the network, the less likely it will be to become overload by changes in demand, making it more flexible. The value of maximum flow is the same as the total weight of the minimum valued cut disconnecting a source node from its sink. A cut is the removal of a set of edges that splits the graph into two parts. In figure 88, the minimum cut is the second from the right with a value of 8.4.

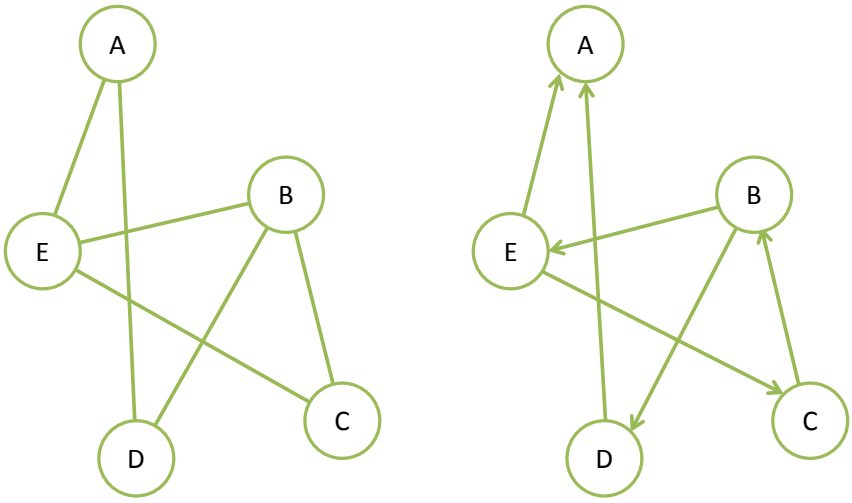


Figure 87: Example graph with connectivity

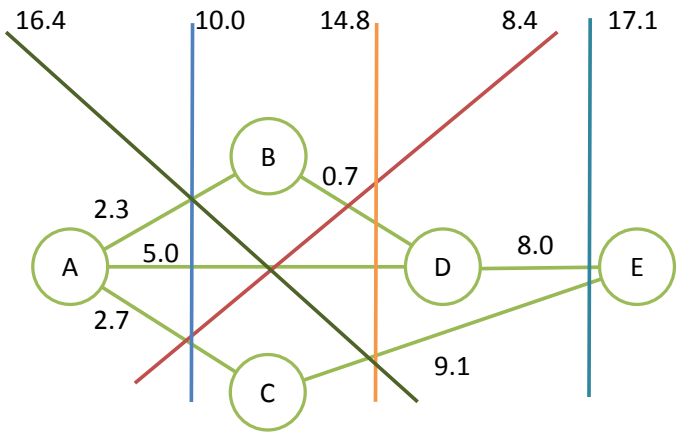


Figure 88: Cuts through a network flow from node A to node E

A.8 Spectral properties

Given that the graph is represented by a matrix, it is trivial to calculate the eigenvalues and eigenvectors of the network if the adjacency matrix is square. The spectrum of a graph is the set of eigenvalues of its adjacency matrix, and the properties of the graph that correspond to the spectrum are called its spectral properties. The spectrum can be calculated by solving the set of equations that result from the relationship $Av = v\lambda$, for adjacency matrix A, with eigenvector v and eigenvalue λ. For a graph with n nodes there will always be n solutions to the system of equations. For a simple graph, the eigenvalues will always be real. However, for a digraph, the adjacency matrix will not be symmetrical, therefore some of the eigenvalues may be complex conjugates of the form $A \pm Bi$

The largest real eigenvalue is called the functional cyclicity and its eigenvector is called the Perron Frobenius Eigenvector (PFE). It is commonly used as an indicator of the number of cycles in the graph.[29] Cycles are paths that start and end at the same node, and like paths they can frequently be used to describe the completion of a task chain or the successful transfer of resources in the system. The Coefficient of network effects (CNE) is that eigenvalue normalized by the total number of nodes and is a measure of the networked effects per node which is an indicator of how well distributed the network is. [35] The sum of the absolute values of all eigenvalues is called the graph energy and is a measure of the dynamic instabilities. [16] Stability, the the ability to maintain a steady state, in a network is important as an indicator of how consistently the network is able to provide the desired capability.

Similar to the incidence matrix is the Laplacian matrix which is defined as

$L = D - A$ where D is the degree matrix and A is the adjacency matrix. The second smallest eigenvalue of this matrix is also called the algebraic connectivity and is an indicator of the graph's connectivity. It's associated eigenvector is called the Fiedler

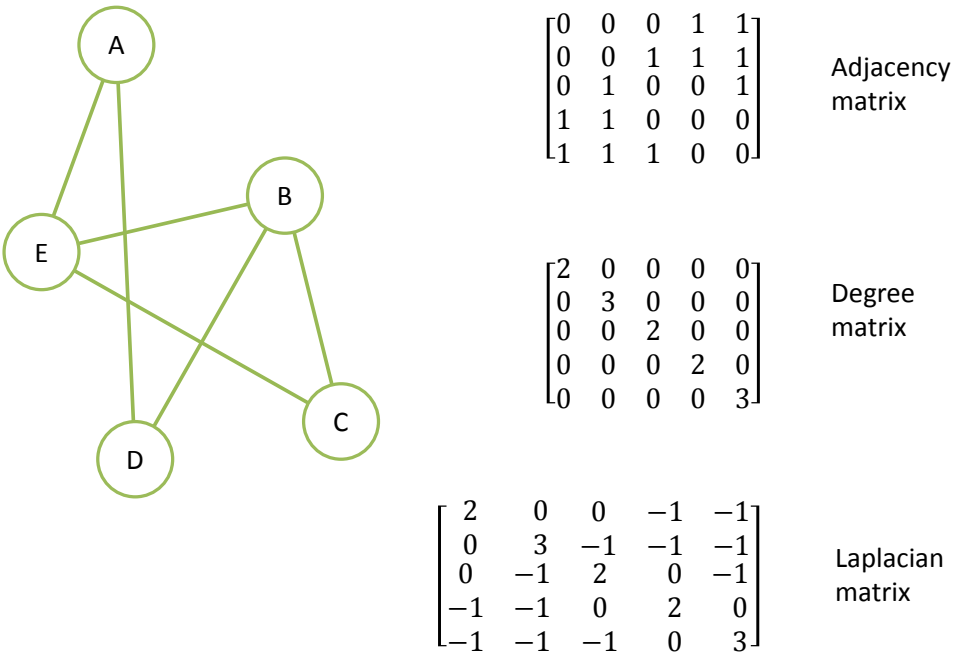


Figure 89: Example graph with adjacency, degree and Laplacian matrices

vector (FV) and is an indicator of how easily the network will synchronize.[120] Synchronization of a network is the case where all the elements are interacting properly and on time and the ability of the network to synchronize is how well does it reach a steady state. A synchronized network will reach stability faster and thus more likely to adapt to changes in a timely manner. [124]

In summary, there are many useful and computationally efficient static properties of a network that have are proven indicators of the actual behavior. One of the advantages of using graph theory is that most network models are deterministic, meaning statistical sampling of stochastic effects is not necessary. It is always a concern that since the system is dynamic and constantly in flux that static measures will correlate poorly. However, it is hypothesized that they will represent the behavior well enough to perform some early trade-offs.

APPENDIX B

NSGA II

The Non-dominated Sorting Genetic Algorithm II is a multi-objective genetic algorithm that was developed to reduce the computational complexity of other non-dominated sorting algorithms, include elitism, and eliminate the reliance on sharing parameters to maintain Pareto front diversity. It was shown to find a "much better spread of solutions and better convergence near the true Pareto-optimal front compared to other multi-objective EAs. [50] Additionally NGPM (NSGA-II Program in Matlab) [145] is an NSGAII software package compatible with Matlab that interfaces very nicely with FLoRA DES

- Reduce the computational complexity of the non-dominated sorting algorithm.

  Older versions of non-dominated sorting algorithms used a simplistic method for determining Pareto rank. Each solution can be compared with every other solution in the population to find if it is dominated. This requires $O(MN)$ comparisons for each solution, where M is the number of objectives. When this process is continued to find all members of the first non dominated level in the population, the total complexity is $O(MN^2)$. This process is repeated for each subsequent non-dominated front. The worst case is when there are fronts and there exists only one solution in each front. This requires an overall $O(MN^3)$ computations.

  NSGA2 uses a faster non-dominated sorting algorithm that requires $O(MN^2)$ calculations instead. "First, for each solution we calculate two entities: 1) domination count n_p, the number of solutions which dominate the solution p , and 2)S_p, a set of solutions that the solution dominates. This requires $O(MN^2)$

comparisons. All solutions in the first non-dominated front will have their domination count as zero. Now, for each solution p with $n_p = 0$, we visit each member (q) of its set S_p and reduce its domination count by one. In doing so, if for any member q the domination count becomes zero, we put it in a separate list Q. These members belong to the second non-dominated front. Now, the above procedure is continued with each member of Q and the third front is identified. This process continues until all fronts are identified. For each solution p in the second or higher level of non-domination, the domination count n_p can be at most $N-1$. Thus, each solution p will be visited at most $N-1$ times before its domination count becomes zero. At this point, the solution is assigned a non-domination level and will never be visited again. Since there are at most $N-1$ such solutions, the total complexity is $O(N^2)$. Thus, the overall complexity of the procedure is $O(MN^2)$." [50]

- Include elitism

  It was shown that elitism improves the performance of multi-objective genetic algorithms by preventing the loss of good solutions. [166, 132]

- Eliminate the reliance on sharing parameters to maintain Pareto front diversity.

  Previous algorithms relied on a sharing function to maintain diversity in solutions along the Pareto front. This function uses a parameter to govern the proximity between two members of the solution population. The problem with this is that the performance of the algorithm would end up being heavily dependent on the users' choice of sharing parameter and require $O(N^2)$ comparisons.

  NSGA2 uses a different method to enforce diversity along the Pareto front. "To get an estimate of the density of solutions surrounding a particular solution in the population, we calculate the average distance of two points on either side of this point along each of the objectives. This quantity $i_{distance}$ serves as an

estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices (call this the crowding distance) The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude. Thereafter, for each objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective functions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance" [50] The algorithm can be shown to have computational complexity $O(MNlogN)$. In this way the GA will favors solutions from the same non-dominated front that occupy less crowded areas.

To summarize, NSGA2 was shown to find a "much better spread of solutions and better convergence near the true Pareto-optimal front compared to Pareto-archived evolution strategy and strength-Pareto EAtwo other elitist MOEAs that pay special attention to creating a diverse Pareto-optimal front." [50] Since the goal of this design space exploration is to sufficiently sample the Pareto front of solutions for multi-objective SoS design problems, NSGA2 promises to be a good choice of algorithm. Additionally, a MATLAB implementation of the algorithm is readily available [145] thereby eliminating the need to develop and validate an algorithm specifically for this study. This is advantageous in that it will integrate easily with the models described earlier that were also implemented in MATLAB.

APPENDIX C

PATH FINDING DFS

Main function

```
function [nPaths all_paths] = PathCountDFS(graph, sources, sinks)
[i, j, w] = find(graph);


nPaths = 0;
all_paths = {};
for ii = 1:numel(sources)
% for each sink->source combination

for jj = 1:numel(sinks)
paths = {};
for kk = 1:numel(find(i==sources(ii)))% find the paths
paths = DFS(paths, [sources(ii)], graph, sinks(jj), kk);
end


all_paths = {all_paths{:} paths{:}};
for kk = 1:numel(paths)% count the paths
if paths{kk}(end) == sinks(jj)
nPaths = nPaths + 1;
end
end
end
end
```

```
end
```

Recursive searching function

```
function [paths] = DFS(paths, path0, graph, sink, n)
[i, j, w] = find(graph);

aa = find(i == path0(end));
%index of first edge leaving the path

if ~isempty(aa) % if the path has not ended
new = j(aa(n));

if new == sink
path1 = [path0 new];
paths{end+1} = path1;

elseif isempty(find(path0 == new, 1))
% if the next node is not a repeat

path1 = [path0 new]; % add it to the path
for ii = 1:numel(find(i == new))
paths = DFS(paths, path1, graph, sink, ii);
% recursively continue the search

end
else
path1 = path0;% otherwise end the path
%              paths{end +1} = path1;
```

```
end
else
path1 = path0;%otherwise return the path
%         paths{end +1} = path1;

end
end
```

REFERENCES

[1] "10 usc 2460: Definition of depot-level maintenance and repair."

[2] *COMPASS STAT User's Manual*.

[3] "Mil-std-1390d military standard: level of repair analysis."

[4] "Navy marks milestone production of key aircraft computer system," August 2012.

[5] "Open automotive alliance." http://www.openautoalliance.net/, 2014.

[6] "Hardware open systems technologies (host) conformant secure network server, navy sbir 2015.1 - topic n151-019," 2015.

[7] "Matlab 'copulafit ' documentation," 2015.

[8] AIR TRANSPORT ASSOCIATION OF AMERICA, *PACKAGING OF AIRLINE SUPPLIESATA Specification No. 300*, 19 ed., July 1996.

[9] ANSI/IEEE, "Recommended practice for architectural description of software-intensive systems," Tech. Rep. 1471-2000, ANSI/IEEE, 2007.

[10] Apple, "Apple carplay." https://www.apple.com/ios/carplay/, 2014.

[11] Ash, G. R., "Dynamic network evolution, with examples from at&ts evolving dynamic network," *IEEE Communications Magazine*, 1995.

[12] Ashton, K., "That 'internet of things' thing," *RFiD Journal*, 2009.

[13] Atkinson, J., "Flexibility: Planning for an uncertain future," *Manpower Policy and Practice*, vol. 1, Summer 1985.

[14] Back, T., "Selective pressure in evolutionary algorithms: a characterization of selection mechanisms," in *1st IEEE conf. on Evolutionary computation*, 1994.

[15] Baeck, T., *handbook of evolutionary computation*. Institute of Pysics Publishing, 1997.

[16] Balakrishnan, R., "The energy of a graph," *Linear algebra and its applications*, vol. 387, pp. 287–295, 2004.

[17] Balb, G. and Chiola, G., "Stochastic petri net simulation," in *1989 Simulation Winter Conference Proceedings*, 1989.

[18] Balestrini-Robinson, S., *A modeling process to understand Complex system Architectures*. PhD thesis, Georgia institute of technology, 2009.

[19] Bankes, S., "Tools and techniques for developing policies for complex and uncertain systems," in *National Academy of Sciences*, vol. 99, pp. 7263–7266, May 2002.

[20] Barros, L. L., "The optimization of repair decisions using life-cycle cost parameters," *IMA Journal of Mathematics Applied in Business & Industry*, vol. 9, pp. 403–413, 1998.

[21] Barros, L. L. and Riley, M., "A combinatorial approach to level of repair analysis," *European Journal of Operational Research*, vol. 129, pp. 242–251, 2001.

[22] Basten, R., *Designing logistics support systems Level of repair analysis and spare parts inventories*. PhD thesis, Beta Research School for Operations Management and Logistics, 2009.

[23] BAUMOL, W. J. and VINOD, H. D., "An inventory theoretic model of freight transport demand," *MANAGEMENT SCIENCE*, vol. 16, 1970.

[24] Beamon, B. M., "Measuring supply chain performance," *International Journal of Operations & Production Management*, 1999.

[25] Benjaafar, S., "Models for performance evaluation of flexibility in manufacturing systems," *International Journal of Product Research*, vol. 32, no. 6, pp. 1383–1402, 1994.

[26] Biltgen, P. T., Ender, T., and Mavris, D. N., "Development of a collaborative capability-based tradeoff environment for complex system architectures," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006.

[27] Black, F. and Scholes, M., "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, pp. 637–654, 1973.

[28] Boehm, G. W., "A spiral model for software development," *IEEE Computer Journal*, vol. 21, pp. 61–72, May 1988.

[29] Bonacich, P. F., "power and centrality: a family of measures," *the american journal of sociology*, vol. 92, pp. 1170–1182, March 1987.

[30] Bondi, A. B., "Characteristics of scalability and their impact on performance," in *Second international workshop on Software and performance*, 2000.

[31] BOUACHERA, T., KISHK, M., and POWER, L., "Level of repair analysis based on genetic algorithm with tabu search," in *Proceedings of the World Congress on Engineering*, 2010.

[32] Box, G. E. and Draper, N., *Empirical Model-Building and Response Surfaces*. Wiley, 1987.

[33] Brown, F., *Engineering System Dynamics: A Unified Graph-Centered Approach*. CRC Press, 2 ed., 2007.

[34] Browne, J., Dubois, D., Rathmill, K., Sethi, S., and Stecke, K., "Classification of flexible manufacturing systems," *The FMS Magazine*, 1984.

[35] Cares, J., *distributed network operations: the foundation of network centric warfare*. newport, RI: Alidade Press, 2005.

[36] Cares, J. R., *Distributed Networked Operations*. Alidade Press, 2005.

[37] Cares, J., "An information-age combat model," tech. rep., Alidade Incorporated, 2004.

[38] Carley, K., "Smart agents and organizations of the future," tech. rep., Carnegie Mellon University.

[39] Carley, K., "Inhibiting adaptation," tech. rep., Carnegie Mellon University, 2002.

[40] Carley, K., *Summary of the NRC workshop on Social Network Modeling and Analysis*, ch. Dynamic Network Analysis. National Research Council, 2003.

[41] CENTER, D. O. T. N. N. H., "C-9 skytrain ii," 2000.

[42] Chen, I. J. and Chung, C. H., "An examination of flexibility measurement and performance of flexible manufacturing systems," *International Journal of Product Research*, vol. 34, no. 2, pp. 379–394, 1994.

[43] Congress, U. S., "10 us code 2464 - core logistics capabilities."

[44] Conway, J., "The game of life," *Scientific American*, vol. 223, no. 4, p. 4, 1970.

[45] Correa, H. L., *Linking flexibility, uncertainty and variability in manufacturing systems: Managing un-planned change in the automotive industry*. Avebury, 1994.

[46] D. P. GAVER, J., "Time to failure and availability of paralleled systems with repair," *IEEE TRANSACTIONS ON RELIABILITY*, 1963.

[47] Dahmann, J., "An implementers view of systems engineering for systems of systems."

[48] Das, S. K., "The measurement of flexibility in manufacturing systems," *The Iraernational Journal of Flexible Manufacturing Systems*, vol. 8, pp. 67–93, 1996.

[49] de Neufville, R. and Neely, J., "Hybrid real options valuation of risky product development projects," *International Journal of Technology, Policy and Management*, vol. 1, pp. 29–46, January 2001.

[50] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 6, 2002.

[51] DeLaurentis, D. A., "Understanding transportation as system-of-systems design problem," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, no. AIAA 2005-123, 2005.

[52] Department of Defense, *Dod modeling and simulation (m&s) glossary.*

[53] Domercant, J. C., *ARC-VM: An Architecture Real Options Complexity-Based Valuation Methodology for Military Systems-of-Systems Acquisitions.* PhD thesis, Georgia Institute of Technology, 2011.

[54] Domercant, J. C. and Mavris, D., "Measuring the architectural complexity of military systems of systems," in *IEEE Aerospace Conference*, no. 1649, pp. 1–16, March 2011.

[55] Dorigo, M., *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, 1992.

[56] Duclos, L. K., Vokurka, R. J., and Lummus, R. R., "A conceptual model of supply chain flexibility," *Industrial Management & Data Systems*, vol. 103, 2003.

[57] Edson, B., "Creating the internet of your things," 2014.

[58] Eswaramurthy, V. and Tamilarasi, A., "Hybridization of ant colony optimization strategies in tabu search for solving job shop scheduling problems," *International Journal of Information and Management Sciences*, 2009.

[59] Evans, J. H., "Basic design concepts," *A.S.N.E. Journal*, pp. 671–678, 1959.

[60] Even, S., *Graph Algorithms.* Cambrige University Press, 2 ed., 2011.

[61] Feibleman, J. and Friend, J. W., "The structure and function of organization," *The Philosophical Review*, vol. 54, no. 1, pp. pp. 19–44, 1945.

[62] Feitelson, E. and Salomon, I., "The implications of differential network fexibility for spatial structures," *Transportation Research Part A*, no. 34, pp. 459–479, 200.

[63] Fiksel, J., "Designing resilient, sustainable systems," *Environmental Science and Technology*, vol. 37, pp. 5330 – 5339, 2003.

[64] Fine, C., *Logistics of Production and Inventory*, ch. Development in Manufacturing Technology and Economic Evalutation Models. Amsterdam, The Netherlands: North Holland, 1990.

[65] Fogel, D. and Michalewicz, Z., *How to solve it: Modern Heuristics*. springer Publishing Company Inc., 2000.

[66] Fogel, D. B. and Ghozeil, A., "A note on representations and variation operators," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 1, pp. 159–161, 1997.

[67] Fonseca, C. M. and Fleming, P. J., "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization," in *Genetic Algorithms: Proc. 5th int. conf.*, 1993.

[68] Fonseca, C. M. and Fleming, P. J., "an overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, pp. 1–16, 1995.

[69] Forrester, J., *Industrial Dynamics*. No. ISBN 0262560011, Cambridge, MA: M.I.T. Press, 1961.

[70] Forrester, J., "System dynamics, systems thinking, and soft or," *System Dynamics Review*, vol. 10, no. 2, pp. 245–259, 1994.

[71] Forsburg, K. and Mooz, H., *A visual explanation of development methods and strategies including the waterfall, spiral, vee, vee+, and v++ models.* Center for Systems Management Inc, 2001.

[72] Forsburg, K., Mooz, H., and Cotterman, H., *Visualizing Project Management: Models and Frameworks for Managing Complex Systems*. John Wiley & Sons, 2005.

[73] Freeman, L. C., "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, march 1977.

[74] Goldberg, D. E., *Genetic algorithms in search, Optimization and machine learning*. Addison-Wesley, 1989.

[75] Gordon, G., "Simulation languages for discrete systems," in *BM Scientific Computing Symposium on Simulation Models and Gaming*, pp. 101–118, 1966.

[76] Gordon, G., *System Simulation*. Prentice Hall, 1969.

[77] Griendling, K., *The Architecture-based Technology Evaluation and Capability Tradeoff Method*. PhD thesis, Georgia Institute of Technology, December 2011.

[78] Gross, J. L. and Yellen, J., *Handbook of graph theory*. CRC Press, 2004.

[79] Gupta, Y. and Goyal, S., "Flexibility trade-offs in a random flexible manufacturing system: A simulation study," *International Journal of Product Research*, vol. 30, no. 3, pp. 527–557, 1992.

[80] Gustafsson, L., "Poisson simulation - a method for generating stochastic variations in continuous system simulation," *Simulation*, vol. 74, pp. 264–274, 2000.

[81] Hamber, B., "T.loads abbreviated systems architecture," in *2001 Winter Simulation Conference*, pp. 749–757, 2001.

[82] Han, E. and DeLaurentis, D., "A network theory-based approach for modeling a system-of- systems," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[83] Hart, A., "Anticipations, uncertainty and dynamic planning," 1940.

[84] Holland, J., *daptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[85] Iacobucci, J. and Mavris, D., "A method for the generation and evaluation of architecture alternatives on the cloud," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pp. 137–142, June 2011.

[86] Iacobucci, J., *Rapid Architecture Alternative Modeling (RAAM): A Framework for Capability-Based Analysis of System of Systems Archittectures*. PhD thesis, Georgia Institute of Technology, May 2012.

[87] Inc, M., "Level of repair analysis (lora)," tech. rep., MTain Inc, 2011.

[88] Jamshidi, M., *Systems of Systems Engineering: Principles and Applications*, ch. chapter 1. CRC Press, 2009.

[89] JMP, A Business Unit of SAS, SAS Campus Drive Cary, NC 27513, *Modeling and Multivariate Methods*, 10.0.2 ed., 2012.

[90] Joint Chiefs of Staff, *Joint Publication (jp 1-02): DoD dictionary of Military and associated terms*, 2009.

[91] Jolliffe, I. T., *Prinicpal Component Analysis*. Springer, 2 ed., 2002.

[92] Kennedy, J. and Eberhart, R., "Particle swarm optimization," *Proceedings of IEEE international conference on neural networks IV*, pp. 1942–1948, 1995.

[93] Kochikar, V. and Narendran, T., "A framework for assessing the flexibility of manufacturing systems," *International Journal of Production Research*, vol. 30, pp. 2873–2895, December 1992.

[94] Koen, B., *Definition of the engineering method*. American Society for Engineering Education, 1985.

[95] Kotov, V., "Systems of systems as communicating structures," *Hewlett Packard Computer Systems Laboratory Paper HPL-97-124*, pp. 1–15, 1997.

[96] Koza, J. R., *Genetic Programming*. MIT press, 1992.

[97] Kozan, K., "Work group flexibility: Development and construct validation of a measure," *Human Relations*, vol. 35, no. 3, pp. 239–258, 1982.

[98] Krackhardt, D. and Carley, K. M., "A pcans model of structure in organizations," in *International Symposium on Command and Control Research and Technology*, June 1998.

[99] Land, A. H. and Doig, A. G., "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[100] Luzeaux, D., Ruault, J.-R., and Wippler, J.-L., *complex systems and systems of Systems engineering*. ISTE Ltd, 2011.

[101] Magee, C. and deWeck, O., "An attempt at complex system classification," Tech. Rep. ESD-WP-2003-01.02, ESD Internal Symposium, 2003.

[102] Malamud, B. and Turcotte, D., "Cellular-automata models applied to natural hazards," *Computing in Science & Engineering*, vol. 2, pp. 42–51, 2000.

[103] Mandelbaum, M., *Flexibility in Decision Making: An Exploration and Unification*. PhD thesis, Department of Industrial Engineering, University of Toronto, 1978.

[104] March, J. and Simon, H., *Organizations*. New York NY: Wiley, 1958.

[105] Marr, J., "Performing the galileo mission using the s-band low-gain antenna," in *Aerospace Applications Conference*, pp. 145–183, IEEE, February 1994.

[106] Matthews, R. and Sweeney, R., "Future airborne capability enviroment," tech. rep., NavAir, 2013.

[107] Mavris, D. N., Bandte, O., and DeLaurentis, D. A., "Robust design simulation: A probabilistic approach to multidisciplinary design," *AIAA Journal of Aircraft*, vol. 36, no. 1, 1999.

[108] McCabe, T. J., "A complexity measure," *IEEE transactions on software engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[109] Morlok, E. K. and Chang, D. J., "Measuring capacity flexibility of a transportation system," *Transportation Research Part A*, vol. 38, pp. 405–420, 2004.

[110] Moses, J., "The anatomy of large scale systems," Tech. Rep. ESD-WP-2003-01.25, ESD Internal Symposium, 2003.

[111] Myers, R. H., Montgomery, D. C., and Anderson-Cook, C. M., *Response surface methodology: process and product optimization using designed experiments*, vol. 705. John Wiley & Sons, 2009.

[112] NASA, NASA Headquarters, Washington, D.C. 20546, *NASA Systems Engineering Handbook*, sp-2007-6105 rev 1 ed ed., December 2007.

[113] NavAir, "Strategic planning imperatives for industrial depot maintenance (2010-2017)."

[114] Nelsen, R. B., *An Introduction to Copulas*. New York: Springer, 1999.

[115] Nilchiani, R., *Measuring Space Systems Flexibility: A comprehensive Six-Element Framework*. PhD thesis, Massachusetts Institute of Technology, September 2005.

[116] of Defense, D., "Department of defense architecture framework, version 2.0 volume ii," tech. rep., Department of Defense, 2009.

[117] of Staff, J. C., *Joint Publication 3-0(JP 3-0) Joint Operations*. 2011.

[118] of the Navy PMA109, D., "Advanced mission computer and displays (amc&d) life cycle cost estimate," tech. rep., Dept of the Navy PMA109, 1998.

[119] Office of the Deputy Under Secretary of Defense for Acquisition Technology, Washington, DC: DOD, *Systems Engineering Guide for Systems of Systems*, version 1.0 ed., August 2008.

[120] Olfati-Saber, R. and Murray, R. M., "Consensus problems in networks of agents with switching topology and time-delays," *IEEE transactions on automatic control*, vol. 49, pp. 1520–1533, September 2004.

[121] PackingPrice.com, "Heavy duty cardboard shipping box price list."

[122] Patrizi, M. D., "Lora and compass," tech. rep., USAMC LOGSA, 2009.

[123] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[124] Pereira, T., "Stability of synchronized motion in complex networks," *arXiv preprint arXiv:1112.2297*, 2011.

[125] Perry, W. L., Button, R. W., Bracken, J., Sullivan, T., and Mitchell, J., "Measures of effectiveness for the information-age navy: The effects of network-centric operations on combat outcomes." RAND, 2002.

[126] peter sandborn, *Cost analysis of electronic Systems*. world scientific publishing, 2013.

[127] Petri, C., *Kommunikation mit Automaten*. PhD thesis, Institut fur Instrumentelle Mathematik, 1962.

[128] Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M., "The bees algorithm," tech. rep., Manufacturing Engineering Centre, Cardiff University, UK, 2005.

[129] Preece, D., *Managing Advanced Manufacturing Technology*, ch. Organizations, Flexibility and New Technology, pp. 355–373. London, UK: IFS Publications, 1986.

[130] Rosenhead, J., Elton, M., and Gupta, S., "Robustness and optimality as criteria for strategic decisions," *Operational Research Quarterly*, vol. 23, pp. 413–431, 1972.

[131] Royce, W. W., "Managing the development of large software systems," in *IEEE WESCON*, pp. 328–338, IEEE, 1970.

[132] Rudolph, G., "Evolutionary search under partially ordered sets," ci-67/99, Dept. Comput. Sci./LS11, Univ. Dortmund, Dortmund, Germany, 1999.

[133] Russel, S. J. and Norvig, P., *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd ed., 2010.

[134] Sage, A. and Cuppan, C., "On the systems engineering and management of systems of systems and federations of systems," *Information , Knowledge , Systems Management*, vol. 2, 2001.

[135] Saleh, J. H., *Weaving time into system architecture: new perspectives on flexibility, spacecraft design lifetime, and on-orbit servicing*. PhD thesis, MIT, Department of Aeronautics and Astronautics, 2002.

[136] Scott, D. M., Novak, D. C., Aultman-Hall, L., and Guo, F., "Network robustness index: A new method for identifying critical links and evaluating the performance of transportation networks," *Journal of Transport Geography*, no. 14, pp. 215–227, 2006.

[137] Sethi, A. and Sethi, S., "Flexibility in manufacturing: A survey," *The International Journal of Flexible Manufacturing Systems*, pp. 289–328, 1990.

[138] Shaw, G. B., *The Generalized Information Network Analysis Methodology for Distributed Satellite Systems*. PhD thesis, MIT, Department of Aeronautics and Astronautics, 1999.

[139] Sherbrooke, C. C., *Optimal inventory modelling of systems. Multi-echelon techniques*. Kluwer, Dordrecht (The Netherlands), 2004.

[140] Shewchuk, J., "A set of generic flexibility measures for manufacturing applications," *International Journal of Production Research*, vol. 37, pp. 3017–3042, September 1999.

[141] Shewchuk, J. and Moodie, C., "Flexibility and manufacturing system design: An experimental investigation," *International Journal of Production Research*, vol. 38, pp. 1801–1822, May 2000.

[142] Slack, N., "Flexibility as a manufacturing objective," *International Journal of Operations & Production Management*, vol. 3, no. 3, pp. 4–13, 1983.

[143] Slack, N., "The flexibility of manufacturing systems," *International Journal of Operations and Production Management*, vol. 7, no. 4, pp. 35–45, 1987.

[144] Slack, N., *The Manufacturing Advantage*. London: Mercury Books, 1991.

[145] Song, L., *NGPM – A NSGA-II Program in Matlab V1.4*. Aerospace Structural Dynamics Research Laboratory, College of Astronautics, Northwestern Polytechnical University, China, 2011.

[146] Sousa-Poza, A., Kovacic, S., and Keating, C., "System of systems engineering: an emerging multidiscipline," *Int. J. System of Systems Engineering*, vol. 1, 2008.

[147] Suarez, "An empirical study of flexibility in manufacturing," *Sloan Management Review*, 1995.

[148] Summers, J. D. and Shah, J. J., "Mechanical engineering design complexity metrics: size, coupling and solvability," *Journal of mechanical design*, vol. 132, pp. 021004–1 – 021004–11, February 2010.

[149] System of Systems Engineering Center of Excellence, sponsored by the Office of the Under Secretary of Defense for Acquisition, T. . L. D. S. S. and Mission Integration, J. F. I. U.-A., *System of Systems Engineering Center of Excellence: SoS Engineering"*. august 2010.

[150] Technologies, C., "Sea basing pilot assessment project," tech. rep., CDM Technologies, San Luis Obispo, CA, September 2006.

[151] Uline.com, "Anti-static shippers."

[152] UPS, *UPS box strength guidlines*.

[153] Upton, D. M., "Flexibility as process probability: The management of plant capabilities for quick response manufacturing," *Journal of Operations Management*, vol. 12, no. 3, pp. 205–224, 1995.

[154] US Department of the Air Force, *AIR FORCE INSTRUCTION 21-101: AEROSPACE EQUIPMENT MAINTENANCE MANAGEMENT*, 2002.

[155] USN, "The united states navy depot maintenance strategic plan 2014-2019."

[156] USN, "Navy training system plan for the an/ayk-14(v) standard airborne computer n88-ntsp-a-50-8822b/a," tech. rep., 2000.

[157] Vanderplaats, G. N., *Multidscipline design Optimization*. 2007.

[158] Venkateswaran, J. and Son, Y., "Distributed and hybrid simulations for manufacturing systems and integrated enterprise," in *Annual Industrial Engineering Research Conference 2004*, 2004.

[159] Villeneuve, F., *A method for concept and Technology exploration of aerospace Architectures*. PhD thesis, georgia institute of technology, 2007.

[160] Volovoi, V., "Modeling of system reliability using petri nets with aging tokens," *Reliability Engineering and System Safety*, vol. 84, pp. 149–161, 2004.

[161] Wainer, G. A., *Discrete-event Modeling and Simulation: A Practitioner's Approach*. CRC Press, 2009.

[162] Weber, R. H. and Weber, R., *Internet of Things*. Springer Berlin Heidelberg, 2010.

[163] Yang, X., *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[164] Zahran, I., Elmaghraby, A., and Shalaby, A., "Evaluation of flexibility in manufacturing systems," in *IEEE International Conference on Systems, Man and Cybernetics*, pp. 49–52, November 1990.

[165] Zelenovich, D., "Flexibility–a condition for effective production systems," *International Journal of Product Research*, vol. 20, no. 3, pp. 319–337, 1982.

[166] Zitzler, E., Deb, K., and Thiele, L., "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.