FRAMEWORK FOR ROBUST DESIGN: A FORECAST ENVIRONMENT USING INTELLIGENT DISCRETE EVENT SIMULATION

A Thesis
Presented to
The Academic Faculty

by

Elise K. Beisecker

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

Georgia Institute of Technology
May 2012

FRAMEWORK FOR ROBUST DESIGN: A FORECAST ENVIRONMENT USING INTELLIGENT DISCRETE EVENT SIMULATION

Approved by:

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

Professor David Goldsman
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Vitali Volovoi
School of Aerospace Engineering
Georgia Institute of Technology

Mr Steve Anderson
Principal Scientist
Naval Surface Warfare Center, Dahlgren Division

Dr. Santiago Balestrini
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 28 March 2012

To my great grandfather, Walter Schiebe

for sharing his love of airplanes and engineering.

ACKNOWLEDGEMENTS

I want to thank my family for their support throughout this process. Thanks to my Dad for teaching me to always have a vision. Thanks to my Mom for her emotional support. To my sisters, Ling and Kaitlyn, for understanding my nerdiness. And my extending family for not asking when I was going to graduate too often.

Thanks to all of my friends for understanding how busy grad school keeps you. Especially all the co-worker I consider close friends. Thanks for giving feedback on conceptual ideas, presentations, and whole sections of this thesis. They helped keep me motivated and on track to graduate.

I want to thank my advisor Dr. Mavris for providing the motivation and insight to make this journey. I would also like to thank Dr. Kirby for getting me involved in research as an undergrad. Thanks to Dr. Nixon for introducing Naval logistics as an exciting area of research and Dr. Balestrini for serving as a technical expert and letting me explore the research options. I would also like to thank Ms Kelly Cooper from ONR for her continued support of my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ABM Agent Based Model

ARG Amphibious Ready Group

BAA Broad Agency Announcement

CONUS Continental United States

CSG Carrier Strike Group

DELAS Discrete Event Logistics Advanced Simulation

DES Discrete Event Simulation

DoE Design of Experiments

ESG Expeditionary Strike Group

INLS Improved Navy Lighterage System

ISB Intermediate Staging Base

JHSV Joint High Speed Vessel

JLOTS Joint Logistics Over the Shore

LCAC Landing Craft Air Cushion

LCAC-R Landing Craft Air Cushion - Replacement

LCU-1600 Landing Craft Utility 1600 class

LCU-2000 Landing Craft Utility 2000 class

LHA/LHD Landing Helicopter Deck

LMSR Large Medium-Speed Roll-on/Roll-off

LPD Landing Platform/Dock

LSD Dock Landing Ship

LSV Logistics Support Vessel

LVS Medium Tactical Vehicle Replacements

MCO Major Combat Operation

MEB Marine Expeditionary Brigade

MEC Medium Exploratory Connector

MEU(SOC) Marine Expeditionary Unit Special Operations Capable

MILP Mixed Integer Linear Programming

MINLP Mixed Integer Non-Linear Programming

MLP Marine Manding Platform

MoE Measure of Effectiveness

MoP Measure of Performance

MPF Maritime Pre-Positioning Force

MRE Meal Ready to Eat

MTVR Medium Tactical Vehicle Replacements

NOLH Nearly Orthogonal Latin Hypercube

NPS Naval Postgraduate School

OMFTS Operational Maneuver From the Sea

ONR Office of Naval Research

ROMO Range of Military Operations

RSM Response Surface Methodology

SB Sea Base

SoS System-of-Systems

SSC Ship to Shore Connector

T-AKE Dry Cargo/Ammunition Ships

T-Craft Transformable Craft

TEU Twenty-foot Equivalent Unit

TSP Traveling Salesman Problem

SUMMARY

The US Navy is shifting to power projection from the sea which stresses the capabilities of its current fleet and exposes a need for a new surface connector. The design of complex systems in the presence of changing requirements, rapidly evolving technologies, and operational uncertainty continues to be a challenge. Furthermore, the design of future naval platforms must take into account the interoperability of a variety of heterogeneous systems and their role in a larger system-of-systems context. To date, methodologies to address these complex interactions and optimize the system at the macro-level have lacked a clear direction and structure and have largely been conducted in an ad-hoc fashion. Traditional optimization has centered around individual vehicles with little regard for the impact on the overall system. A key enabler in designing a future connector is the ability to rapidly analyze technologies and perform trade studies using a system-of-systems level approach.

The objective of this work is a process that can quantitatively assess the impacts of new capabilities and vessels at the systems-of-systems level. This new methodology must be able to investigate diverse, disruptive technologies acting on multiple elements within the system-of-systems architecture. Illustrated through a test case for a Medium Exploratory Connector (MEC), the method must be capable of capturing the complex interactions between elements and the architecture and must be able to assess the impacts of new systems). Following a review of current methods, six gaps were identified, including the need to break the problem into subproblems in order to incorporate a heterogeneous, interacting fleet, dynamic loading, and dynamic routing. For the robust selection of design requirements, analysis must be performed

across multiple scenarios, which requires the method to include parametric scenario definition.

The identified gaps are investigated and methods recommended to address these gaps to enable overall operational analysis across scenarios. Scenarios are fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables. Introducing matrix manipulation into discrete event simulations enables the abstraction of sub-processes at an object level and reduces the effort required to integrate new assets. Incorporating these linear algebra principles enables resource management for individual elements and abstraction of decision processes. Although the run time is slightly greater than traditional if-then formulations, the gain in data handling abilities enables the abstraction of loading and routing algorithms.

The loading and routing problems are abstracted and solution options are developed and compared. Realistic loading of vessels and other assets is needed to capture the cargo delivery capability of the modeled mission. The dynamic loading algorithm is based on the traditional knapsack formulation where a linear program is formulated using the lift and area of the connector as constraints. The schedule of demands from the scenarios represents additional constraints and the reward equation. Cargo available is distributed between cargo sources thus an assignment problem formulation is added to the linear program, requiring the cargo selected to load on a single connector to be available from a single load point.

Dynamic routing allows a reconfigurable supply chain to maintain a robust and flexible operation in response to changing customer demands and operating environment. Algorithms based on vehicle routing and computer packet routing are compared across five operational scenarios, testing the algorithms ability to route connectors without introducing additional wait time. Predicting the wait times of interfaces based on connectors en route and incorporating reconsideration of interface to use

upon arrival performed consistently, especially when stochastic load times are introduced, is expandable to a large scale application. This algorithm selects the quickest load-unload location pairing based on the connectors routed to those locations and the interfaces selected for those connectors. A future connector could have the ability to unload at multiple locations if a single load exceeds the demand at an unload location. The capability for multiple unload locations is considered a special case in the calculation of the unload location in the routing. To determine the unload location to visit, a traveling salesman formulation is added to the dynamic loading algorithm. Using the cost to travel and unload at locations balanced against the additional cargo that could be delivered, the order and locations to visit are selected. Predicting the workload at load and unload locations to route vessels with reconsideration to handle disturbances can include multiple unload locations and creates a robust and flexible routing algorithm.

The incorporation of matrix manipulation, dynamic loading, and dynamic routing enables the robust investigation of the design requirements for a new connector. The robust process will use shortfall, capturing the delay and lack of cargo delivered, and fuel usage as measures of performance. The design parameters for the MEC, including the number available and vessel characteristics such as speed and size were analyzed across four ways of testing the noise space. The four testing methods are: a single scenario, a selected number of scenarios, full coverage of the noise space, and feasible noise space. The feasible noise space is defined using uncertainty around scenarios of interest. The number available, maximum lift, maximum area, and SES speed were consistently design drivers. There was a trade-off in the number available and size along with speed. When looking at the feasible space, the relationship between size and number available was strong enough to reverse the number available, to desiring fewer and larger ships. The secondary design impacts come from factors that directly impacted the time per trip, such as the time between repairs and time to repair. As

the noise sampling moved from four scenario to full coverage to feasible space, the option to use interfaces were replaced with the time to load at these locations and the time to unload at the beach gained importance. The change in impact can be attributed to the reduction in the number of needed trips with the feasible space. The four scenarios had higher average demand than the feasible space sampling, leading to loading options being more important. The selection of the noise sampling had an impact of the design requirements selected for the MEC, indicating the importance of developing a method to investigate the future Naval assets across multiple scenarios at a system-of-systems level.

CHAPTER I

MOTIVATION

1.1 Shift in Naval Operational Concept

Sea Power 21, the Navy's operational vision for the 21st century, identifies three fundamental concepts for continued operational effectiveness: Sea Strike, Sea Shield, and Sea Base. Sea Strike is the ability to project precise and persistent offensive power from the sea; Sea Shield extends defensive assurance throughout the world; and Sea Base enhances operational independence and support for the joint force [64]. The intent of the Sea Base is to develop a maneuverable, scalable collection of platforms that enable power projections from the sea [16]. Seabasing is defined in the DoD Dictionary as the deployment, assembly, command projection, reconstitution, and reemployment of joint power from the sea without reliance on land bases within the operational area [96].

The emphasis has shifted to a need for the Navy to project power ashore, deriving from a movement away from land based operations. The United States has already run into difficulties securing bases on foreign soil and political factors may continue to reduce the availability, strengthening the drive toward sea based projection [180]. Political pressures against granting basing rights to U.S. forces is strong, such as the inability to obtain permission from Saudi Arabia or Turkey leading up to the invasion of Iraq [51]. Admiral Moore and General Hanlon stated "Sea Basing exploits the operational shift in warfare from mass to precision and information, employing the 70 percent of the earth's surface that is covered with water as a vast maneuver area in support of the joint force" [183].

The Sea Base construct provides a potential framework for projecting power ashore

with minimal forces ashore, minimizing the need to build up logistical stockpiles ashore. This includes the ability to assemble, equip, project, support, and sustain those forces without reliance on land bases within the Joint Operations Area [16]. The concept for Seabasing is shown in Figure 1. Removing the need for diplomatic arrangements to assure forward basing coupled with forward positioning enables immediate employment [52]. The potential benefits of Seabasing include [15]:

1. Assuring access worldwide for military operations
2. Enhanced forward-defense posture
3. Improvement in immediate response capability
4. Rapid initiation of joint command and control
5. Very rapid transition from crisis to joint forcible entry
6. A greater degree of force tailorability and scalability

The composition of the Sea Base is not established and is intended to be tailorable, but will include distributed forces including carrier strike groups (CSGs) , expeditionary strike groups (ESGs), combat logistics force ships, Maritime Pre-Positioning Force (MPF) platforms, and potentially, high-speed support vessels [51]. Potential vessels seen as part of the Sea Base are given in Figure 2. The Sea Base's contributing elements do not operate in isolation but are part of a logistical chain from production in the continental United States (CONUS) to use by the warfighter in theater. The Sea Base connectors are contributing elements to the logistical function and need to be analyzed as part of a larger throughput process [79]. This leads to the recommendation by the National Research Council that a comprehensive systems analysis of Seabasing ships and connectors needs to be undertaken at a macro level to validate the requirements, such as range, speed, and capacity for cargo and personnel [15].

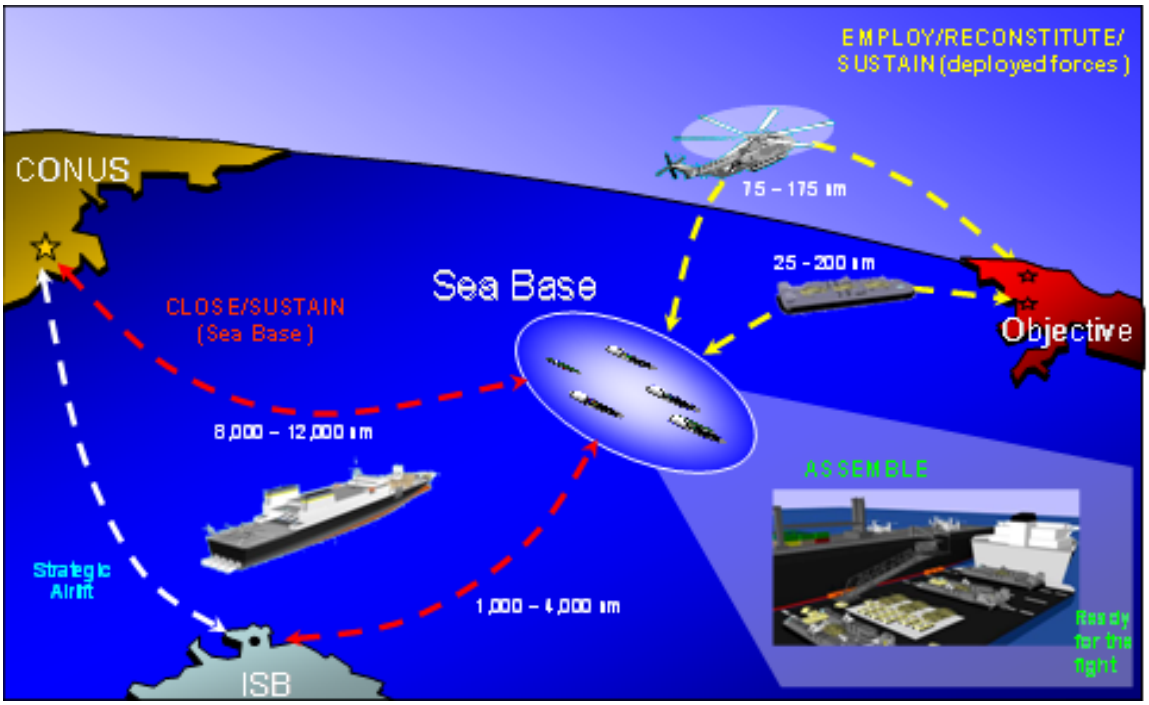


Figure 1: Seabasing Concept [65]

1.2 Challenges of a Sea Base

The Defense Science Board identified Seabasing as a critical capability. One of the greatest challenges to the Seabasing architecture is the transfer of cargo, troops, and equipment from the Sea Base to the objective [9]. The traditional iron triangle, mandating a trade off between speed, range and payload, limits the current alternatives such that no one craft can meet all of the objectives of the Sea Base. The major stressers in meeting the objectives are the required stand-off distances, high sea state transfer capabilities, desire for insertion during one period of darkness, and the need for over-the-beach (feet dry) delivery [79]. These stressers should drive the selection of technologies.

A new long range, medium lift connector needs to be developed to address breaking the iron triangle and be designed to meet the requirements in order to capitalize on the promised Seabasing capabilities. To meet the needs of the Seabasing concept, it

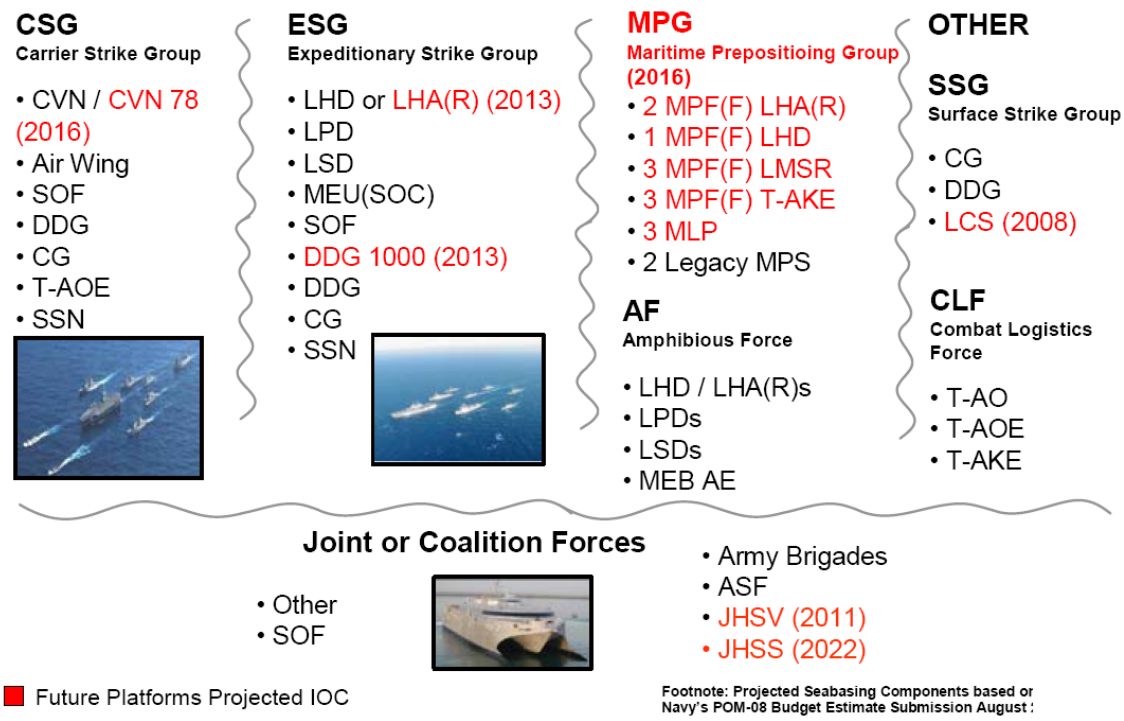


Figure 2: Joint Seabasing Components [126]

has been suggested that future surface connectors must be able to operate in three modes [14]:

1. Fuel efficient, good sea-keeping mode

2. High-speed shallow water mode

3. Amphibious mode to traverse sand bars and mud flats

These three modes cannot be achieved by any existing vessel and formed the starting requirements outlined in the Office of Naval Research's (ONR) Broad Agency Announcement (BAA) soliciting proposals for a prototype demonstrator of a Transformable Craft (T-craft) . The basic requirements highlight an inportant gap in existing connectors. A new type of connector is needed based on the need for a self deployed asset that can deliver intact units with options for interface and transfer of cargo. This thesis will explore the modeling needs and requirements definition for

this Medium Exploratory Connector (MEC).

The Sea Base has been identified as a complex system-of-systems (SoS) by the Defense Science Board [9]. A SoS is defined by the Defense Acquisition Guidebook as "a set or arrangement of systems that results from independent systems integrated into a larger system that delivers unique capabilities" [19]. The Systems Engineering Guide for Systems of Systems recognizes the importance of incorporating system interdependencies in systems acquisition [139].

To measure the capabilities of the future Navy fleet, the conceptual connector must be modeled as it will operate together with the future legacy fleet. To evaluate the capabilities of the MEC as it operates in a Seabasing operation, a model must be flexible enough to incorporate the ability to vary the scenario variables since the Seabasing scenario is not fixed. Any proposed model must incorporate performance and operations of the future legacy fleet and the interactions between these vessels and the MEC, including which vessels can interface with each other and the methods of cargo transfer. Legacy simulations are generally not capable of representing a nonlinear battlespace or one filled with a variety of operating units and lack the flexibility to integrate new elements on short notice at reasonable cost [12].

1.3 Importance of Modeling

The difficulty in assessing the impact of a new vessel's capabilities at the SoS level is that the effects of said vessel are truly complex. Systems-of-Systems tend to qualify as complex systems because (1) they are composed of a large number of interdependent systems, whose (2) interactions are nonlinear, and (3) their overall behavior cannot be predicted by studying the parts in isolation. It is not the goal of this research to join the extensive list of references in which a definition of complexity is attempted, but simply to illustrate the fact that when studying large-scale architectures, or Systems-of-Systems and in particular, how individual capabilities impact their overall behavior,

the exercise becomes a study in complexity.

Looking at a variety of fields of study: telecommunications, logistics, computers, transportation, work flows, information systems, or production, the global effects caused by a local decision are unknown due to the large number of parts and connections. The overall architecture, connection, and local behaviors of the components are easily described but the global behavior requires a model based evaluation method [202].

The overall challenges of Seabasing when viewed in the context of expeditionary maneuver warfare and the evolving strategies of Operational Maneuver from the Sea, the Ship to Objective Maneuver, and Marine Corps Distributed Operations, exhibits a multiplicity of requirements which are often contradictory. Numerous solutions have been proposed as physical elements of a future Sea Base, however, these solutions rely upon a particular vision making a comparison across uniform Measures of Effectiveness (MoE) difficult. To perform a fair comparison of adding new vessels or other elements to a Sea Base, it is necessary to examine MoEs at the SoS level.

1.4 Observations

The following list distills observations about the challenges and gaps of modeling a Sea Base concept and designing a new connector such as the MEC.

- Due to the large number of interacting assets (ships, aircraft, helicopters, ports, landing spots etc.) in a realistic simulation of a Sea Base, many system level models must be created and managed. The development of these models and the simulation of multi-system interactions will require new methods in data handling and system modeling.

- Traditional ship design has been based on required performance metrics. A system-of-systems level focus introduces the capability to investigate factors

such as interfaces between assets and logistics chain, in conjunction with doctrine, policy, and tactics. These factors may have more impact on the design than traditional measures of effectiveness but could not be explored with current methods.

- The design of a system-of-systems involves the interaction of a large number of heterogeneous systems, requiring an approach that addresses the modeling of large-scale problems with both continuous and discrete design variables. The complexity of systems that consist of a variety of heterogeneous subsystems demands a decomposition of the task into a set of smaller, more manageable design problems.

- Campaign-level simulations are traditionally hard-coded for an example scenario and are rarely parametric, flexible, or transparent to the user. While these demonstrations are helpful for visualizing the given scenarios, this format is not conducive to trades at the system and subsystem level across multiple scenarios.

- The significant amount of data generated by the design of multiple assets in diverse vehicle classes is nearly impossible to comprehend. An approach is needed that captures the physics of the problem, allows the decision maker to visualize the results, and facilitates real-time design in a cross scenario, system-of-systems framework.

1.5 Research Goals

The design of complex systems such as ships, submarines, aircraft, and missiles, in the presence of changing requirements, rapidly evolving technologies, and operational uncertainty continues to be a challenge. Furthermore, the design of future naval platforms must take into account the interoperability of a variety of heterogeneous systems and their role in a larger system-of-systems context. To date, methodologies

to address these complex interactions and optimize the system at the macro-level have lacked a clear direction and structure and have largely been conducted in an ad-hoc fashion. Traditional optimization has centered around individual vehicles with little regard for the impact on the overall system. A key enabler for reduced cost and cycle time is the ability to rapidly analyze technologies and perform trade studies using a system-of-systems level approach.

The objective of this work is a framework that can quantitatively assess the impacts of new capabilities and vessels at the systems-of-systems level. This new methodology must be able to handle a diverse fleet of vessels and vehicles while capturing technological developments acting on multiple elements within the system-of-systems architecture. It must also be capable of capturing the complex interactions between elements of the architecture and must be able to assess the impacts of new systems such as the MEC.

For this new quantitative assessment, the research has been broken into two portions. The first is to evaluate existing processes and benchmark their strengths and weaknesses. This will require the evaluation of tools and techniques that can be leveraged in this new methodology and will serve to determine the feasibility of a proposed new solution. The second portion will be the implementation and testing of the proposed methodology on an example system.

In Chapter 2, a brief exploration of logistic modeling techniques is presented. This includes methods and techniques that have been applied to large logistics systems and the strengths and shortcomings of these options. Chapter 3 develops the gaps identified in the previous chapter and develops observations and research questions that lead to assertions and hypotheses. The challenges associated with these hypotheses are discussed in Chapter 4 including the connection between the research questions and hypotheses. The research plan and experiments to demonstrate the hypotheses are given in Chapter 5. Chapters 6 through 9 detail the development and testing

of the model elements followed by a detailed model description and validation of a portion of the model capabilities. Chapters 13 through 15 describe the application to the example system. This work concludes with a summary of interesting observations and contributions.

CHAPTER II

LITERATURE REVIEW AND BENCHMARKING

2.1 Traditional Methods

Vehicle routing has been an area of study since 1959 when Danzig and Ramser published "The Truck Dispatching Problem" [67]. The traditional vehicle routing problem is a fleet of identical vehicles with set demand for each node which does not exceed the capacity of a single vehicle and where each customer is visited once by a single vehicle [106]. Many solution algorithms exist for solving this class of problems directly and some work has been done to expand the types of systems modeled.

The traditional supply chain problem defines a network of facilities that procure materials, transform, and distribute them with the aim to optimize locations, including site and number, production, products to be produced at locations, inventory levels, and transportation [184]. Mixed Integer Non-Linear Programming (MINLP) has been used to optimize these problems, such as selecting the number and location of facilities in a closed loop manufacturing process (manufacture, distribute, consume, recycle, usable part remanufactured) [71]. These problems aim to minimize the overall cost and are developed using mean production rates. The behavior of the system can be fully captured by a set of constraints and flow description equations. Changing the network or constraints involves redeveloping the set of equations and rerunning the optimization thus uncertainty can only be handled by generating solutions to distinct scenarios. Many MINLP or Mixed Integer Linear Program (MILP) problems of realistic size are too large to be solved exactly. Problems handling production planning, which includes solving selection, batching and loading problems simultaneously, can be decomposed into submodels, pooling machines to solve the selection and batching

and isolating the loading problem [97].

Another limitation of using MILP or MINLP is the difficulty in handling a heterogeneous fleet. Teypaz, Schrenk, and Cung [178] explored application to a heterogeneous fleet with the demand at each node being equal. They used a non-limited set of vehicles to satisfy a demand in a set period of time by choosing the transportation network of the most profitable vehicle first moving to the least profitable. Each type of vehicle was filled as a max flow problem. The limitation of this approach is that each type of vehicle was considered independently in the construction of routes and schedules.

The Sea Base concept is a time domain problem beyond the setting of a time period to complete the delivery. The assault phase could have a set constraint, but resupply is a continual process. Each phase could be considered to have the goal to minimize the time required (make span), such as the work by Zegordi, which considers a two-stage supply chain where suppliers are located in different geographical regions and products are transported from suppliers to a manufacturing company [199]. This problem has been shown to be an NP-hard structure with distribution in a single area using one type of vehicle. NP-hard problems can not be solved exactly but a large number of heuristics such as genetic algorithms, evolutions strategies, and partical swarms can be applied [195, 148]

2.1.1 Classification of Problem

The Sea Base problem discussed in this work exceeds the capabilities of traditional supply route and vehicle loading problems. Larson discusses enhancements to traditional vehicle routing that can be used to model dynamic problems such as just-in-time logistics with the focus on time-dependent data [109]. A problem can be classified as dynamic using the following criteria [149]:

1. Time dimension is essential

2. Problem may be open-ended
3. Future information may be imprecise or unknown
4. Near-term events are more important
5. Information update mechanisms are essential
6. Resequencing and reassignment decisions may be warranted
7. Faster computation times are necessary
8. Indefinite deferment mechanisms are essential
9. Objective function may be different
10. Time constraints may be different
11. Flexibility to vary vehicle fleet size is lower
12. Queuing considerations may become important

The Sea Base problem meets all of these criteria, especially the open-ended nature of the problem, the focus on near-term events, and queuing considerations. The suggested solving method for this type of problem is to use traditional optimization algorithms for short term prediction and repeat the optimization at set time intervals [150].

2.1.2 Gaps in Traditional Methods

A model of the Sea Base could be developed using traditional methods and the time domain considerations could be included using a dynamic vehicle routing method, but there are gaps that remain in the type of information of interest to this study. The first limitation of classical scheduling theory is the assumption that each machine can process at most one job, processing speeds do not change in time, and process

times are fixed and known [83]. With a Navy cargo ship, there are distinct options for handling a connector, which do not require the same process time, and multiple connectors may be able to interact at the same time. For example, an LMSR may be able to use its starboard ramp and rear ramp deck at the same time.

Evaluation using optimization is limited to deterministic or handling uncertainty through the use of limited scenario based evaluation. Changes in the number of nodes or network structure would require developing new constraint equations. The incorporation of a heterogeneous fleet is limited to deferring to the most profitable option first and solving for each type independently. There would be no interactions between types of connectors, which is unrealistic since connectors would need to use the same cargo transfer options.

One additional limitation, which is unique to the Sea Base construct is that the location of the storage facilities may not be fixed. The cargo ships of the Sea Base would start at some prepositioned location and travel into theater. In this, they are acting as both a transport and cargo supply asset, a scenario that can not be handled by traditional methods of vehicle routing and supply chain analysis.

2.2 Simulation Based Methods

Mathematical methods are not flexible enough for this large scale system, since analytical solutions of dynamic systems often require simplifying assumptions [90]. This inability to model the increased complexity needed for realistic studies using mathematical tools has lead to a growth in simulation, defined as [100]:

> the practice of building models to represent the existing real-world systems, or hypothetical future systems, and of experimenting with these models to explain system behavior, improve system performance, or design new systems with desirable performances.

Three simulation techniques commonly used for logistics modeling are examined for applicability to this problem. The investigation is not meant to be an exhaustive description of simulation, but a highlight of techniques used for large scale systems.

2.2.1 Expected Value

The concept of expected value is the time to complete an operation can be estimated using a series of movements and estimated queues. One such model is the Joint Expeditionary Logistics Operations (JELO) model, developed at the Naval Postgraduate School (NPS) , which calculated an expected value estimation of total operation time including close, transfer, and deployment [42]. This Excel-based model allows for development of scenarios defining the troop and equipment movement to reach the Sea Base. Each scenario is a series of movements between locations and waiting periods for equipment to arrive, mate, load, or unload. The time required to complete a movement is calculated using the distance covered and the transit speed. The deployment time is calculated by a separate movement model which calculates the time needed to move a set number of units of a type of cargo, which is dependent on the type of cargo, number of connectors available, number of operational spots available, and the distance cargo is transported. The time is for a single type of connector and cargo thus the overall deployment schedule must be developed by the user knowing the mix of commodities and the schedule for deployment. The total deployment time and any necessary operational pause time are added to the transit and assembly time to yield an overall mission time.

Expected value models are useful in their simplicity and the ability to examine a wide variety of scenarios and ease of use. Unfortunately, it only calculates a single type of connector and cargo type so scheduling must be computed separately. The loading queue calculations are estimated based on usage factors and other queues, such as unloading, are not considered. This modeling technique does not provide the

needed degree of detail.

2.2.2 Agent Based

Agent Based Models (ABM) are ideal when modeling interaction between autonomous, decision making entities with the following characteristics [43]:

- Individual behavior is nonlinear and can be characterized by discrete decisions, thresholds, if-then rules,or nonlinear coupling.
- Describing discontinuity in individual behavior is difficult with differential equations. For example, if a logistics officer orders parts in batches, he may have a threshold for making parts requests (rather than continuously demanding replacements for parts used).
- History matters. Path-dependence, lagging responses, non-Markovian behavior, or temporal correlations including learning and adaptation are applicable to the system.
- Averages are not good enough. Under certain conditions, small fluctuations in a complex system can be amplified, so that the system is stable for incremental changes but unstable to large perturbations.

ABM have been used for force-on-force simulations and Tripp expands its use for military logistics when the logistics domains are distributed and have decentralized organization and control [182]. Roorda, et al. developed a conceptual framework to incorporate multiple levels of decision making including fundamental business changes, supply chain management, market interactions, and logistics with agents representing the shipper, receivers, and carriers [154]. This framework is conceptual and would require a great deal of effort and further development to become an operational model.

ABM is not suitable for the system of interest and the type of answers desired. The individual behavior of the connectors is well defined and the decision making does not

need to be autonomous. ABM would be a good investment if detailed analysis of the onshore demand is of interest and could be used to monitor supply levels and make logistics requests. In this case, ABM is not matching the nature of the problem in that individual behavior is not as interesting and the objects actions can be described as a process instead of a set of decisions [44].

2.2.3 Discrete Event Simulation

Discrete Event Simulation (DES) has a strong track record in improvements of production processes [166]. A discrete system refers to the nature of the changes in behavior with respect to time, where discrete changes occur in finite quanta or jumps, and continuous systems change continuously over time. This type of simulation can give variances, extremes, and time series in addition to performance averages [147]. The underlying idea comes from everyday experience where system states and events are discrete, and includes queuing models such as requiring some sort of service, which is restricted in a way that not all concurrent requests can be answered at the same time, for example, waiting for an available teller at a bank, a seat in a cafeteria, or to cross the street [202]. DES problems embody the following concepts [77]:

1. Work - items, jobs, or customers seeking service
2. Resources - provider of service
3. Routing - collection of required services
4. Buffers - waiting area for work awaiting service
5. Scheduling - pattern of resource availability
6. Sequencing - order resources provide service (e.g. first come first serve)
7. Performance - overall system measure

The Sea Base embodies these seven concepts, with the connectors as the work, seeking to interface for cargo as resources. They are routing between pick-up and drop-off locations for cargo and the number of completed trips and total cargo delivered can be measured. Queuing models are able to describe systems with resource allocation and sequences of operations on a much higher level [202], especially when compared to agent based or mathematical methods without loosing information about the individual objects.

2.2.4 Existing DES Models

DES is a conceptual framework with many types of applications and the development of the application determines the capabilities of the model. A handful of models exist using discrete event simulation to model a Sea Base type operation, demonstrating the variety of types of application. The development, advantages, and disadvantages of each simulation are discussed below.

2.2.4.1 NPS Model

This model was developed as part of a year long class project in the Wayne E. Meyer Institute of Systems Engineering (WEMISE) at the NPS focusing on expeditionary warfare [141]. This DES was developed in Extend and accounts for accumulating, assembling, deploying, and sustaining expeditionary forces. A Marine Expeditionary Brigade (MEB)-sized force is built from CONUS and forward-deployed forces and assembled at a set location. Once assembled the forces are moved to the launching areas and deployed in scheduled waves of air and surface connectors. This flow of ships and material is illustrated in Figure 3.

Sustainment has connectors moving cargo to shore as well as bringing additional supplies from offshore bases. This model also includes a probability of the connector failing to reach the shore with the load lost and the probability of the asset needing repair which includes modeling down time.

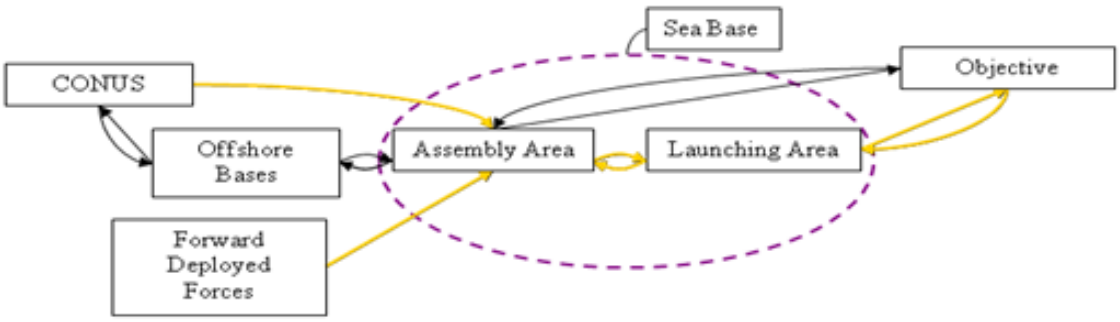


Figure 3: NPS Model Flow

Advantages

- Includes many types of ships and air assets
- Includes possibility of connector vessels needing repairs but requires a good estimate of the probability of failure for system
- Provides time plots for supplies on shore
- Includes constant consumption model which allows for calculation of sustainment needs

Drawbacks

- With the large number of assets, the model becomes computationally complex and lacks transparency
- Adding a vessel requires modifying many sub-process to duplicate model steps
- Limited loading options are pre-set and must be prioritized with no check that the loading schedule is realistic
- Sea Base and Shore are resource pools, individual vessels are not tracked as to loading spots or cargo on board
- Changing scenario would require redevelopment of model

2.2.4.2 T-LoaDS/C-LoaDS

Tactical Logistics and Distribution System (TLOADS) is an in-depth model built in Extend and SDI Industry Pro to model of the transportation of supplies from the Sea Base to end use nodes [144]. A scenario is defined by selecting and placing standard force compositions, forming a series of distributed nodes. TLOADS provides a detailed analysis of consumption of supplies at the nodes, which order supplies from supplier node. The node to supply the materials is determined along with the appropriate containers and vehicles to use for the shipment and the time to make the shipment [143]. In depth data needed includes vehicle load capabilities, vehicle maintenance profiles and material handling equipment distribution between nodes and capabilities. C-LoaDS add the analyses of movement of material within a ship, including the distribution of stock in the stowage areas, assignment of spots for incoming transports and the flow of material between the transports and the stowage areas [143]. The problem is that any modification of the included test case led to software failure and many key elements were not implemented according to the model support files.

2.2.4.3 CDM Humanitarian

CDM Technologies, Inc. developed a set of in depth humanitarian missions of the T-Craft and generated data using a set of candidate solutions [58]. These T-Craft solutions were compared to the performance of a larger number of ship to shore connectors (SSCs) and cargo helicopter. In their work, the schedule was developed by a separate algorithm and serves as an input to the DES. The algorithm starts with vessel 1 and schedule until operating time is reached, then move on to the next until the maximum number of vessels is reached. Landing zone 1 must be fulfilled before starting landing zone 2 and so on, which can cause queuing at landing zones. If a demand is not met within the time period, it expires and is not transported

on sequential day (aka each day is considered independently). In addition, the Sea Base was assumed not to limit operations so vessels are loaded immediately upon arrival. Vessels are loaded until the area in the maximum number of containers or the payload capacity for that concept (including movers and trailers) is met In this case Medium Tactical Vehicle Replacements (MTVRs) , Logistics Vehicle Systems (LVSs) with containers, and tractor-trailers with containers were considered in three different analysis. It was assumed types of cargo can be mixed within a container.

The limitations of this model are that variations on two scenarios were considered and the scenario can not be easily changed because scenario definition requires a large amount of detailed information. Each vessel was analyzed separately, so a heterogeneous fleet was not considered. Queuing to unload was considered in the scheduling algorithm, but the transfer of cargo at sea was not incorporated.

2.2.4.4 CLF Scenario Analysis Tool

This model was developed by Morgan to measure the impact of adding a high speed vessel (HSV) to current Combat Logistics Force (CLF) assets [130]. Two scenarios based on Major Combat Operation (MCO) were analyzed but the scenario is flexible and is defined by a set of nodes and arcs representing shipping points and usable connections. The ship performance information is an input as well as the location, commodities capability, and use rates. Customer ships are generators of logistics requirements, which are fulfilled by shuttles such as the HSV. Ports serve as supply points but strategic resupply to ports is not handled. This model contains many important aspects, such as dynamic scenarios defined by nodes and arcs and ship starting points, but is not broad enough for application to the Sea Base.

2.2.4.5 RAND Sea Base Model

The Assessment Division of the Office of the Chief of Naval Operations (OPNAV N81) asked the RAND Corporations National Defense Research Institute to examine

the capabilities of the Sea Base concept [52]. This model has a set architecture and vessel definition. Adding additional vessels, even of the same type, would require code modification. Three analysis cases were run including air assets and some considerations for Landing Craft Air Cushion (LCAC) and Joint High Speed Vessel (JHSV) . Queuing at the Sea Base is considered but there is no considerations of reliability. Cargo delivered is in terms of vehicles, supplies, and number of personnel and mixed cargoes are not allowed, with the model written so any surplus sustainment capability going to dry stores. This answers the questions, can the operation in question be sustained in a certain amount of time? The time periods are independent and no sorties are carried over. This model has the queuing needed but lacks the flexibility in the scenario, including varying the number of vessels present.

2.2.4.6 JWARS

The Navy Warfare Development Command used JWARS, the Joint Warfare System, developed by MITRE Corp, to examine the logistical demands of supporting a MCO from a Sea Base [17]. JWARS is a constructive simulation that includes 1) an explicit three dimensional battlespace, 2) the effects of terrain and weather, 3) logistically constrained force performance, 4) explicit representation of key information flows, and 5) perception based command and control [125]. It requires detailed data input and forces decision makers to choose the exact force structure and scenario. A variety of types of assets can be used and the characteristics, interaction, and performance of these assets must be fully defined, including attrition probabilities.

2.3 Hybrid Methods

Discrete Event Simulation provides a useful framework for modeling the Sea Base but leaves unanswered some of the issues, such as selecting routing and loading connectors. Klemmt, et al. [103], suggests creating a hybrid of object-based simulation models and mathematical optimization, wrapping processes, such as job scheduling,

in an optimizer. Schlegel, et al. [166], recognizes the challenge of using optimization of a full model but suggests using mathematical optimization for sub-problems. This technique distinguishes between process and operational design problems and the scenario variables become inputs to the optimization. DES tends to focus on the sequential issues such as deadlock and blocking, while optimization, such as assignment or dispatching, focus on performance assessment, assuming the absence of conflict [87].

Optimization and simulation can be used in a hybrid method bringing together the strengths of each. Optimization can solve subproblems involving selection and dispatching while overall flow can be controlled by the simulation. The optimizers can use current model state to make routing and sequencing selections such as cargo to load from what location and where to bring it to.

2.4 Need for a New Method

The field of vehicle routing and supply chain management is a well developed field with over fifty years of research. Elements from this field are key to developing a usable analysis method for measuring a new connectors impact on the operation of the Sea Base. The discrete event simulation framework will be able to provide a basic foundation for the queuing and processes associated with a Sea Base type operation. Mathematical methods can provide a formulation for solving sub-problems not traditionally handled by DES. The breakdown into component problems that can be solving using known and documented techniques as been lacking.

There are limited applications of the needed components of the Sea Base model, but gaps in the application and methods remain which will serve as the basis for the work presented in this thesis. Previous work has demonstrated one or a few of these concepts but fail to generalize the models to a level where all these concepts can be incorporated into a single analysis. These gaps are:

- Breakdown of modeling problem into component
- Parametric scenarios
- Heterogeneous, interacting fleet
- Dynamic loading
- Dynamic routing
- Analyzing design requirements across multiple scenarios

In general, a specific scenario or a handful of scenarios are developed to serve as an input to the analysis model. There is a need for parametric scenarios including not only different area in the world, handled in the CLF Scenario Analysis Tool, but different types of operations. Most of the tools in existence have a preset, hard coded fleet. The performance and interactions of the vessels and other assets were preset and part of the structure of the code. This includes pre-sets of what the vessels and other assets can carry and in what combination. There is no mechanism for changing the cargo load-outs on connectors based in the scenario of interest. The cargo needed and location of resupply does not effect the processes completed by the connectors, so their routing is set in advance and not dynamic with changes in the current state of the operation.

The needs observed in the previous chapter can not be fulfilled by existing models and methods so new techniques for incorporating these elements are needed. The specific questions deriving from these observations and the proposed work needed are discussed in the next chapter.

CHAPTER III

PROBLEM DEFINITION

The observations and research questions are broken down into six topics that define the gaps in current methods and models, listed in 2.4. The first gap involves the breakdown of the modeling problem into elements that can be solved through modeling and mathematical approaches. The overall problem is large and can not be solved by existing methods but elements can be decomposed and addressed. One such element to be addresses is the need for parametric scenarios, including the need for dynamic loading. The need for a heterogeneous, interacting fleet demands a new data handling method. The routing must be based on the vessels and cargo available and not pre-set. By closing these gaps, the design requirements for a new connector can be analyzed across multiple scenarios.

3.1 Problem Decomposition

Observation: The complexity of systems demands a decomposition of the task into a set of smaller, more manageable design problems

Research Questions:

1. Different types of vehicles will complete different processes but are there common elements that can be abstracted?

2. Which sub-problems are abstractable and can be dealt with as separate problems?

3. Can the sub-processes discussed be treated as individual problems?

4. Will abstracting sub-processes decrease the effort required to integrate a new

asset?

Assertion: Interface selection, loading, and routing sub-problems are abstractable

Experiments: Experiments will be posed for each sub-problem

3.2 Parametric Scenarios

Observation: Model must capture multiple scenarios

Research Questions:

1. What is the minimum amount of information needed to define a scenario?
2. Can this information be defined by a set of variables allowing a single model for several scenarios?

Assertion: Scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables

3.3 Fleet Size and Interactions

Observations:

- A system-of-systems focus must consider interfaces between assets and fleet mix in addition to traditional asset performance
- Treating vessels as cargo objects will require new methods in data handling and modeling

Research Questions:

1. How can the interfaces between assets be treated as design variables and not a pre-set option?
2. How can changing the fleet mix not involve changing the structure of the model?

3. Can cargo nodes be treated as objects including the ability to track cargo and interface use?

Hypothesis: Introducing matrix formulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduce the effort required to integrate new assets

3.4 Loading Problem

Observation: Dynamic cargo loading enables parametric scenarios

Research Questions:

1. Can mathematical optimization techniques be used for local level decisions?

Hypothesis: Knapsack loading is an efficient and robust approach for solving the loading sub-problem

3.5 Routing Problem

Observation: Routing is needed so the supply chain is a product of the assets present

Research Questions:

1. Can routing decision be made using parameters tracked in the model?
2. Can the logistics chain formulation be a byproduct of the selected mix of assets?

Hypothesis: Matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing

3.6 Analysis to Identify Design Drivers

Observation: An approach is needed that facilitates design in a cross scenario, system-of-systems framework

Research Questions:

1. What design parameters of a new connector are key to improving the overall performance of the heterogeneous system?

2. Does a common set of parameters exist across multiple scenarios?

3. Should scenarios be defined using continuous variables or as a discrete selection? Is this a necessary model design choice?

4. How can these parameters be identified when scenarios are uncertain?

Hypothesis: Feasible scenario robust analysis identifies the design drivers for a range of scenarios

3.7 Proposed Contributions

1. Incorporating linear algebra into DES will establish a more flexible construct

   (a) Enable the addition of new assets without changing the structure of the model or existing assets

   (b) Routing, loading, and cargo selection will be abstracted as seperate problems, removing them from the asset process

   (c) Abstracting subproblems will reduce the amount of effort required to modify these choices or to add assets

2. Flexibility to define a variety of scenarios

   (a) Model structure is not modified by changing scenario

   (b) Dynamic routing allows the logistics network to be a product of the scenario and available assets, not established by the model formulation

3. Apply robustness principles across feasible scenarios to identify design drivers for systems that will be used in a variety of mission types

CHAPTER IV

CHALLENGES OF HYPOTHESES

This chapter details the challenges and needs associated with each hypothesis identified in the previous chapter.

4.1 Scenario Definition

This work asserts: Scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables

It is necessary to be able to define scenarios in such a way that changing the scenario does not change the formulation of the model. The goal is to be able to define a scenario based on a set of parameters. This is enabled by the incorporation of the matrix manipulation. The scenario should be left as broad as possible to represent a wide range of possible future scenarios including user inputs for the number of vessels and vehicles involved. Where these assets start and are resupplied further defines the scenarios. Landing zones are treated as objects, so their number and properties must be set. For this model, it is assumed that the Sea Base, or at least the vessels modeled here, are at the same general distance from the shore. A key aspect to defining the scenario is to define the needs onshore with a day by day schedule. A generic cargo vector of minimally transportable units must be defined and the need is mapped to this vector as well as the priority for transport.

4.2 Augmenting DES for Large Models

Expansion of hypothesis: Introducing matrix manipulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduce the

effort required to integrate new assets

4.2.1 Limitations of Traditional Resource Management

Limited resources within a DES model are managed as a set number of a resource. For example, a factory may have a limited number of processing machines, so when a product reaches that step in the process, it must wait until a resource is available, reserve a set number of that resource and after a set amount of time, release those resources. The difficulty with this model is that there are multiple options to interface and transfer cargo. To select an interface option, all possible options must be explored and the fastest option should be selected. This construct creates a series of if-then statements to find open options, followed by selecting the one with the minimal wait time. This if-then construct had to be created for each type of vessel, since the possible connections and times are different for each type of connector. An additional challenge of modeling this large scale system is that the resources are not in a central pool. The resources exist on individual objects and are not independent. For example, a LMSR has a side ramp and stern ramp, which may or may not be used at the same time. In order to consider the cargo aboard individual ships and other vehicles, it is necessary to treat them as individual objects and to know which cargo object the connector chooses to interface with. Thus, cargo selection must be a component of resource selection. The introduction of linear algebra principles enable resource management for individual elements and will speed selection of an interface connection as well as speeding the addition of new types of connector, cargo objects, and supply ports.

4.2.2 Incorporating Marix Manipulation

4.2.2.1 Petri Nets

There are several ways to model discrete event systems including queuing models and petri nets. Queuing models are a system of inputs, queues, and limited services while a petri net is a bipartite directed graph [92]. Petri Nets have been recognized

for their ability to model discrete event systems and have been expanded through the use of colored petri nets to support the information aspects needed for DES [89, 135]. Matrices have been used in coordination with discrete event modeling in the form of abstracting Petri Nets to matrix manipulation [175]. The transitions within the Petri Nets are controlled using matrices [128, 196, 112]. The matrices are used to describe the state spaces [54] as well as the transition rules. Giordano establishes that matrices can be used as a controller including tracking where and which material handling equipments should be dispatched in a section of a factory. This dispatch controller is important where in-advance planning is inapplicable. The dispatch controller can be implemented at a supervisory and operational level where rules can be established with associated costs [88]. The operational level control includes route selection and conflict resolution [87]. Giordano goes on to discuss that if tasking can be described as a set of if-then rules based on the current perception of the environment, then a matrix-based controller can rigorously represent these rules [86]. In addition to material handling equipment, matrix based controllers have been used in mobile robotics control. DiPaola identifies a matrix based controller as modular and easily reconfigurable for changes in mission or hardware configurations while being fast and intuitive [68].

Matrix control theory has been expanded beyond petri nets to vector discrete event systems (VDES). Li and Wonham [114] developed VDES to model systems using vector addition where the states of the system are maintained in a vector and the transitions form matrices. This framework can be visualized as a petri net except in the most generalized case. VDES is used to develop state feedback controllers for a manufacturing plant. The concept has been applied to a limited number of situations including operational safety in [164] for a reactive controller.

4.2.2.2 Application to DES

Turk identifies that DES may be too slow to model complex networks since the complexity grows superlinearly, and prefers system dynamics since capabilities are calculated through matrix manipulations [185]. By using linear algebra to replace traditional if-then constructs, DES will be able to handle complex large scale system models without sacrificing evaluation time. The traditional construct builds if statements, repeated for each type of interface usable by a connector, then the selection of the fastest interface. Instead, the interface options are stored as a matrix with the options available to a connector stored as a vector. Using row comparison, the available connections are identified and when this resulting matrix is multiplied by the wait times, the minimum of the matrix identifies the fastest available loading time. In addition, this identifies which ship or other cargo object has the interface available.

4.2.2.3 Additional Capabilities Enabled by Linear Algebra

This construct is flexible enough to handle a variable number of assets in the simulation and changes in asset properties. One purpose of this simulation is to be able to vary the fleet mix present, which would change the number of assets overall. This is a strength of linear algebra in that the sizes of the matrices and vectors can be dynamically sized and the process would be unchanged and the calculation time is minimal. The only constraint is that the sizes of the matrices must be consistent if dealing with several. The flexibility to change object properties during the simulation, such as role enables assets to change function. For example, a cargo ship may serve as a connector traveling from a supply point, but becomes a cargo object upon arrival at the Sea Base. In traditional resource management, the resource would have to be added to a specific pool at that location where this framework will handle this automatically. This also allows for modeling objects that become available over time and functions

that modify objects. How this formulation is used in the DES is detailed in the next chapter.

4.3 Challenges of Loading Problem

Development of the hypothesis: Mixed Integer Linear Programming is the most efficient and robust approach for solving the loading sub-problem

Realistic loading of vessels and other assets is needed to capture the cargo delivery capability of the modeled mission. Maximum cargo delivery could easily be calculated by tracking the total number of trips completed by each type of connector. But, in reality, connectors will not carry their maximum weight and area since individual loads will cube out (volume limited) or weigh out. The Sea Base should not be considered a generic source for cargo since each individual cargo ship has its own cargo and interface options. The use of matrices solves the handling of cargo assets as individual assets, leaving the subproblem of selecting the cargo to load on an individual connector.

The idea of preset loadings, as seen in [141], was dismissed because of the desire to be able to handle multiple scenarios without changing the structure of the model. The matrix manipulation described above is expandable to include which cargo objects have the cargo needed by the connector. If the cargo needed can be expressed as a vector, this vector can be compared to the cargo available vectors for each cargo objects. The resulting matrix is compared to the interfaces available, identifying which interfaces are available on cargo objects that have the cargo needed. This expansion of the matrix manipulation solves the selection of a connection point with cargo but does not handle the identification of what cargo needs to be brought.

The cargo to be loaded needs to be selected based on some priority and would be loaded until the maximum weight or area constraint was met for the connector in question. Carroll and Isaacson discuss dynamically route cargo to theater by

prioritizing the cargo using different prioritization methods and determining shipment until total air and sea capability are surpassed [55]. This idea that cargo could be prioritized by the user is combined with the weight and area constraints to determine the cargo to be loaded in the next connector.

The cargo to be transported is defined in terms of minimally transportable units with associated weights and areas. The weights and areas form vectors of information about each type of unit. It was assumed that the user could provide a cargo schedule based on the order these units should be delivered to shore. Using the order, weights, and areas along with the weight and area constraints of the connector determine what should be carried on that connector. An example mathematical construct will be presented in Section 8.1.1.

This method had several shortcomings, which will be shown in Section 8.2. If the next unit that needs to be carried is too large to fit on connectors available, these connectors will not be dispatched until a connector large enough to carry this unit arrives. In addition, item mixing is minimal because similar units are listed together and this does not load the connectors efficiently as a smaller unit may have fit on board, but would not be loaded since it is not next on the list.

4.3.1 Loading as a Knapsack Problem

Karabuk suggests using mathematical optimization for a portion of the problem and uses the example of treating the loading of individual trucks in a transportation system model as a knapsack problem [98]. The objective of a knapsack problem is to find the most valuable selection of possible items that satisfy the weight constraint [200]. There are two types of problems, linear where a fraction of an item can be packed in the knapsack and integer, where an item is either completely packed or not at all [53]. Since the cargo is expressed as minimally transportable units, an entire unit must be carried, defining the problem as an integer knapsack. The application

of a knapsack algorithm to ship loading is suggested by Dano [66].

The loading problem will be formulated as a mixed integer linear program (MILP). The MILP to be solved follows where P_i is a prioritization of the cargo units, $weight_i$ is the weight of the unit, and $area_i$ is the area required to transport that unit. The maxlift and maxarea are the constraints for the connector of interest. $Needed_i$ is the total number of a type of unit to be delivered so the total load does not exceed the maximum demand.

$$Maximize \sum_i P_i x_i$$

subject to:

$$\sum_i weight_i x_i \leq maxlift$$

$$\sum_i area_i x_i \leq maxarea$$

$$x_i \geq 0 \text{ for all i}$$

$$x_i \leq needed_i \text{ for all i}$$

4.4 Challenges of Routing Problem

Reasoning behind the selection of hypothesis: Matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing

Supply chain configuration is the selection of which units (suppliers) to include and the links among units, in this case which cargo supply points (cargo ships or supply ports) and which connectors to use. Dynamic routing would allow for a re-configurable supply chain to maintain a robust and flexible operation in response to changing customer demands and operating environment [59]. Chandra and Grabis define robustness as the ability to handle a loss of supplier and flexibility to choose transportation channels. The dynamic routing scheme must be able to handle the loss

or gain or a supplier or connectors and be able to choose the connector or to choose which route the connector should travel. This will allow the model to not have a pre-set logistics flow but be represented as a network of possible nodes and connections. Stradtler calls this process orientation, involving the allocation of activities to members by using key performance indicators to reveal weaknesses, especially at interfaces between members, which may lead to reallocation of activities [170]. The difference is illustrated in Figure 4 and Figure 5. The first figure shows each connector having a fixed pair or load and unload locations, aside from initial positioning. The second figure shows the supply chain as a network where the connection selected depends on the demand and current status of the model.

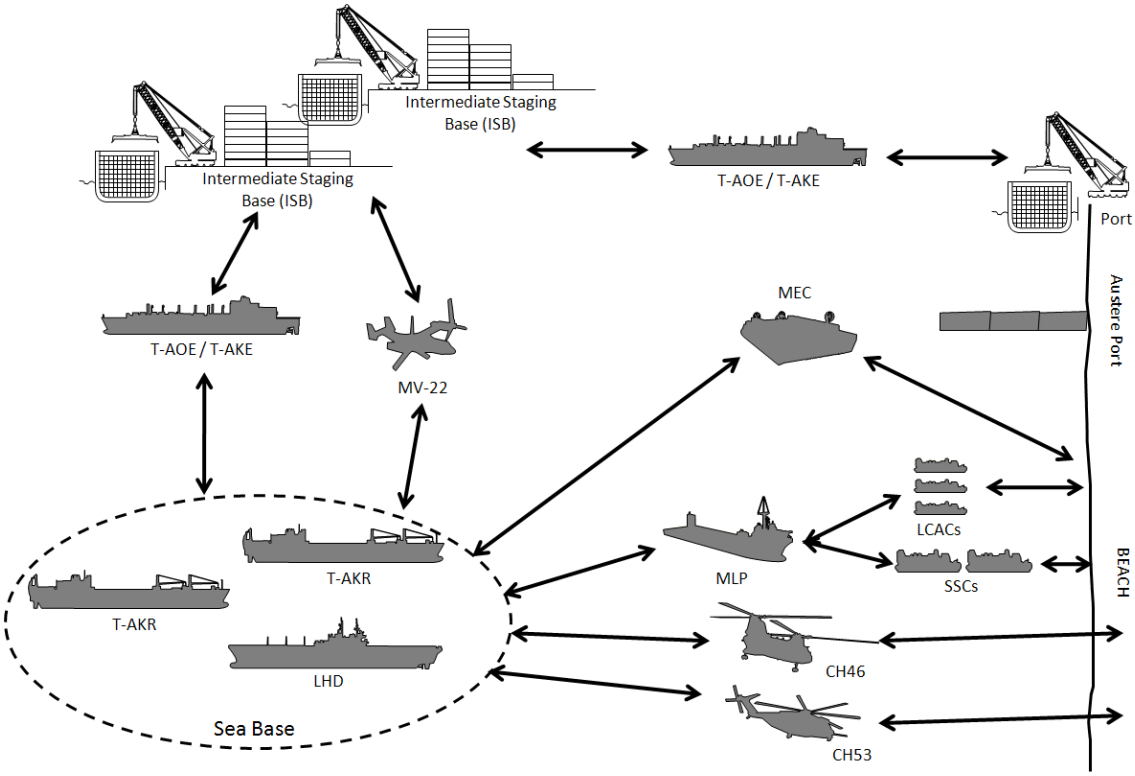


Figure 4: Routing without Dynamic Algorithm

The key performance parameter of interest here is the forecasted time to complete a resupply mission. This incorporates the travel time to reach a location as well as the expected queue time. Once a connector completes a delivery, it must select its

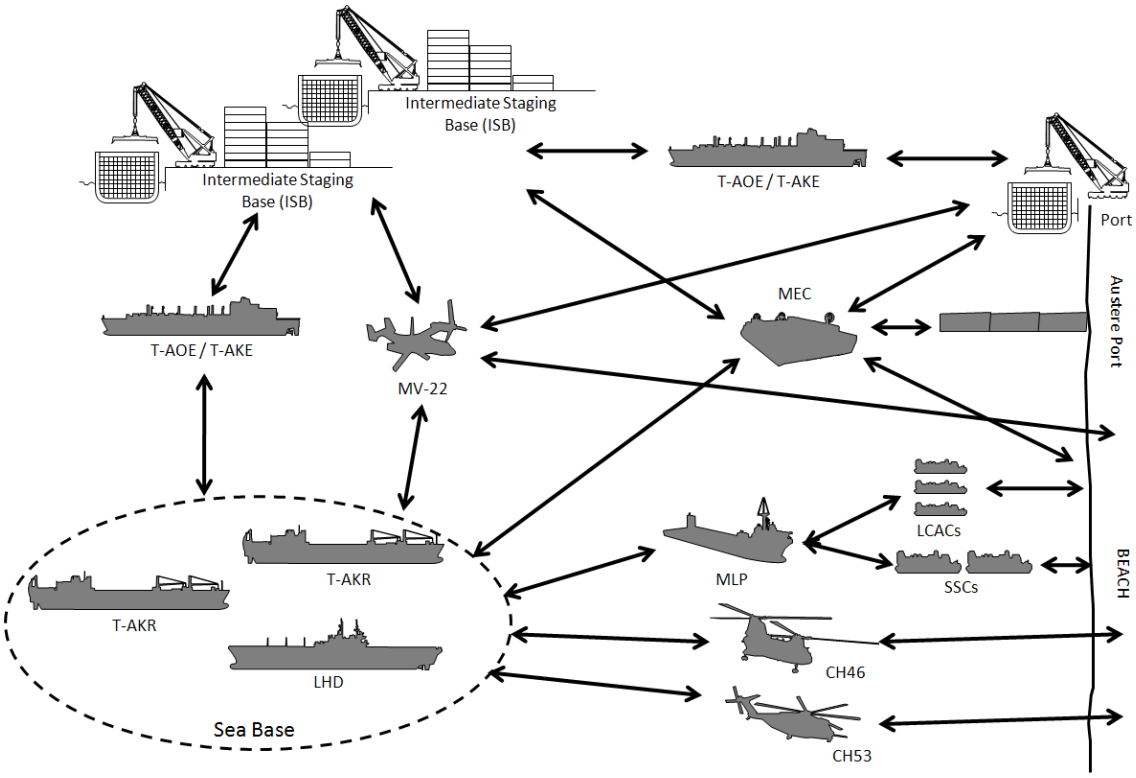


Figure 5: Routing with Dynamic Algorithm

next set of cargo load and unload locations. This selection is performed by estimating the time it would take to complete the potential missions. For each potential unload point, for example, the conceptual connectors could bring cargo to the shore or reload the Sea Base, the model predicts the cargo to be carried, which interface would be selected for reload and the load time is added to the predicted queue. The location and destination pair that has the shortest travel and queuing time is selected as the next reload point. Since this is predicted and does not include uncertainties, such as required repairs, the connectors join the first in first out queue when they arrive at the cargo load point.

4.5 Robustness and Sensitivity Analysis

Genesis of the ideas behind the hypothesis of: Feasible scenario robust analysis identifies the design drivers for a range of scenarios

Any Navy asset will have to operate in a variety of types of operations across the Range of Military Operations (ROMO) . The variety of scenarios can be modeled with the proposed process, but the question remains, how can the design be analyzed across the scenarios of interest? Sensitivity analysis can identify driving parameters in a single scenario, using the Pareto plot, developed by Dr. Joseph M. Juran to identify the driving factors that effect quality [33]. The key design factors may not be the same for every scenario, so a method is needed to analyze at least a family of scenarios. In the case of design, this analysis is key since the designer will have to live with the choice of design and be judged against its future performance [23]. Optimization techniques have been used with discrete event simulations models [174], but how can the design be optimized for several scenarios?

Where sensitivity analysis identifies the sensitivity of a solution to changes in input data, robust design formulates designs that are less sensitive to model data [133]. In this case, it is desirable to find a design that is less sensitive to changes in scenario. Dr Taguchi defines robustness as [176]:

> the state where the technology, product, or process performance is minimally sensitive to factors causing variability (either in the manufacturing or user's environment) and aging at the lowest unit manufacturing cost

Taguchi measured robustness in terms of a signal-to-noise ratio and used a loss function to capture deviations from a target value. In this case, robustness will be related to the ability of the fleet to deliver the cargo needed in a timely fashion. Robust design is not a new technique and has been paired with discrete event simulations. Sanchez details a method for trading off performance mean and variability by examining the expected loss function [161]:

$$E(loss) = c[\sigma^2 + (\mu - \tau)^2]$$

Where c is a scaling constant and τ is a target state. An experimental design is

selected incorporating the decision and noise factors. Metamodels can be created to represent the mean and standard deviation, which are combined into the expected loss function to identify robust configurations. This process has been used even for queuing systems and is highlighted as being flexible and efficient. The framework for a robust design that follows is detailed by Sanchez [162, 160]:

1. Select the performance measures
2. Specify a loss function
3. Identify the factors
4. Plan the experiment
5. Analyze the results
6. Refine the metamodels
7. Select the best process design

This process was used by Scheibe [165], with decision factors including T-Craft design parameters and the number of connectors present. Within a humanitarian scenario modeled in ARENA, the noise variables included deck use, number of shore spots, probability of hit and sink,and attrition rate. The outputs used were time to complete, percent cargo delivered, and portion of craft destroyed. This lead to recommendations on the number of T-Craft and lift capability. Cason [56] used robust design for the design of a Vertical Takeoff and Landing Unmanned Aerial Vehicle (VTUAV) using an agent based model of an infantry platoon conducting patrolling operations in an urban environment. The noise variables focused on the unknown enemy characteristics and the design variables were at the vehicle level, including sweep and speed.

Applying robust design to the problem examined in this thesis will require mixing level or variables not seen in the example work. A limited number of vessel design

variables and scenario variables were included in Scheibe's work. Building on Scheibe's work, Cason included many more vessel level variables. The design variables would be the design parameters of the MEC including size and speed as well as design choices such as which interfaces the MEC can use. The noise variables would include the scenario definition variables such as cargo to deliver, distances, other vessels and vehicles that would be present, and number of beach spots.

4.5.1 Feasible Scenario Robust Analysis

As discussed in 2.2.4, most current models examine a single or limited number of scenarios, and this could be used to analyze the scenarios of interest. Each relevant scenario would be fully defined and serve as a single evaluation point. The robust design would be completed using the scenario as a categorical variable to switch between selected inputs. There are variable that are a combination of design and scenario, such as the number of landing spots. This depends on the area of operation, including the geographical quality of the shore, local populace, and degree of distribution, yet this is also coupled to the craft climbing ability. More beach may be accessible to craft that can climb a beach with a higher angle if the geographic characteristics are the limiting factor. The impact of variables that may indicate scenario and vehicle impacts would be lost with categorical scenarios.

On the other hand, the scenarios could be fully parametric. Each variable that serves to define the scenario is treated as an independent continuous or discrete variable. This would completely cover any possible scenario but it would also lead to unrealistic scenarios, such as bringing in only tanks and tents. This may lead to a biasing of the results based on unrealistic scenarios. The feasible design space can not be expected to be regular and some techniques for identifying the feasible region is detailed by Bates, et al [36].

It is necessary to sample enough of the scenario space to encompass the uncertainty

in future operations without sampling unrealistic space. The scenario space is a large hyperspace with one dimension for each scenario variable, but there are correlations between the dimensions. For example, a tank company would not be deployed without some support units. It is proposed that feasible scenarios will be closely related to current scenarios. This is illustrated in Figure 6, where the gray area represents the entire design space and four traditional scenarios are represented by black circles. The light boxes are the feasible scenario space. The size of the feasibility box in each dimension would vary with the type of variable. Variables such as the amount of each type of cargo to be delivered are bounded and other variables, such as number of landing location, cover the entire design space. The sampling for generating data should fall in the feasible scenario space, so an advanced sampling method is necessary. Sampling techniques will be examined for applicability to a segmented design space.

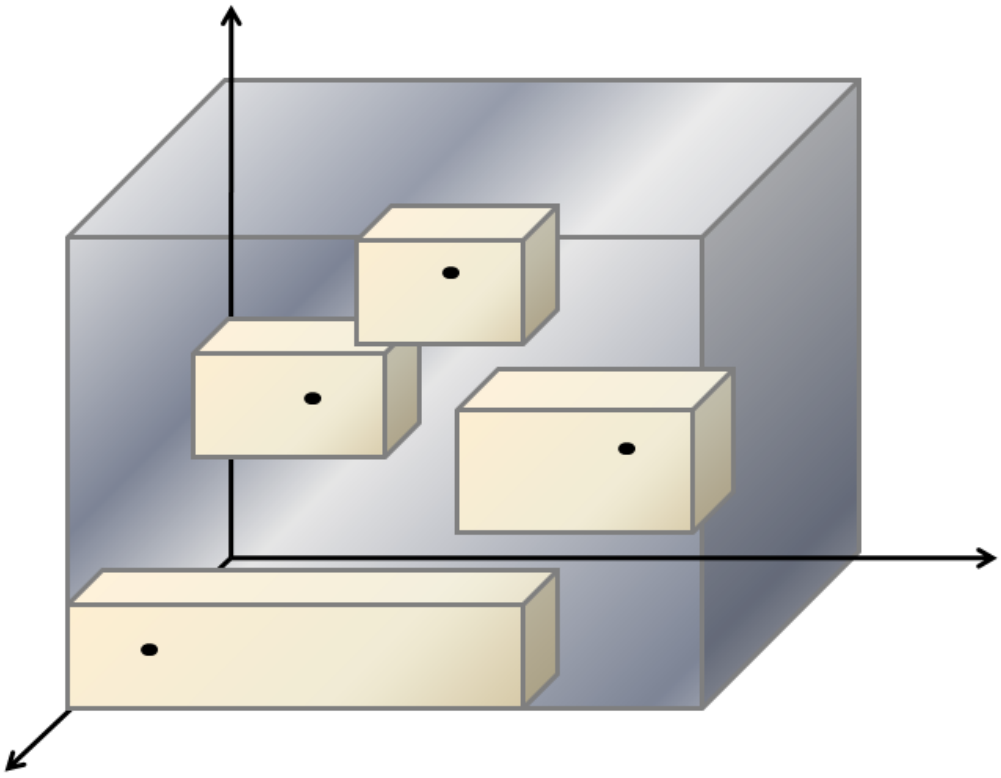


Figure 6: Feasible Scenario Sampling

Methods have been developed for dealing with robustness across multiple scenarios. Vommi and Seetala [192] suggest using a weighted robustness factor, proportional to the liklihood of the scenario, to examine multiple scenarios. Linderman and Choo suggest that there is a diminishing return on adding additional scenarios, where beyond an 'optimal' number, additional scenarios will not improve the results [118]. This work proposes experimenting to identify if the solutions differ when analizing a handful of scenarios, feasible portions of the design space, and the entire design space.

There are variables that are not equally possible within the design range, which could be replaced with realistic distributions. Work is being performed by Hyun Seop Lee at the Aerospace Systems Design Laboratory to develop distributions of the distances from current US supply bases to locations around the world. He randomly samples location onshore and identifies the closest supply base, developing a distribution for the distance to travel. These distances can be weighed by instability factors, such as political freedom and likelihood of natural disasters, resulting in a weighted distribution of distances to travel. These distance distributions could be used as a sampling bases in the robustness analysis.

The next chapter will go into further detail and develop the experiments to address the challenges of these hypotheses.

CHAPTER V

RESEARCH PLAN

The observations and research questions are broken down into six topics that define the gaps in current methods and models.

1. Breakdown of modeling problem into component
2. Parametric scenarios
3. Heterogeneous, interacting fleet
4. Dynamic loading
5. Dynamic routing
6. Analyzing design requirements across multiple scenarios

This results in the following assertions and hypotheses:

Assertions:

1. Interface selection, loading, and routing sub-problems are abstractable
2. Scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables

Hypotheses:

1. Introducing matrix formulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduce the effort required to integrate new assets

2. Knapsack loading is an efficient and robust approach for solving the loading sub-problem

3. Matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing

4. Feasible scenario robust analysis identifies the design drivers for a range of scenarios

The assertion requirements for the first assertion are decomposed into the sub-problem hypotheses. The remaining assertion and hypothesis are furthered in Chapter 4, resulting in the experiments detailed in this chapter.

5.1 Parametric Scenarios

A small number of representative scenarios have been developed to represent military and humanitarian missions. It is important to include the full range of military operations as the Sea Base is seen by the US Marine Corp to have a wide range of applications [18]. The Range of Military Operations (ROMOs) include combat and non-combat elements as seen in Figure 7. It is necessary to define a cargo vector generic enough to capture this range of operations.

5.1.1 Assertion Requirements

1. Identify and quantify scenarios relevant to MEC design

2. Develop generic cargo vector including minimally transportable units

5.2 Model Development

The inclusion of matrix manipulation has been incorporated as demonstrated in the model description as well as using integer programming for the loading problem. The dynamic routing problem must be incorporated into the matrix formulation through

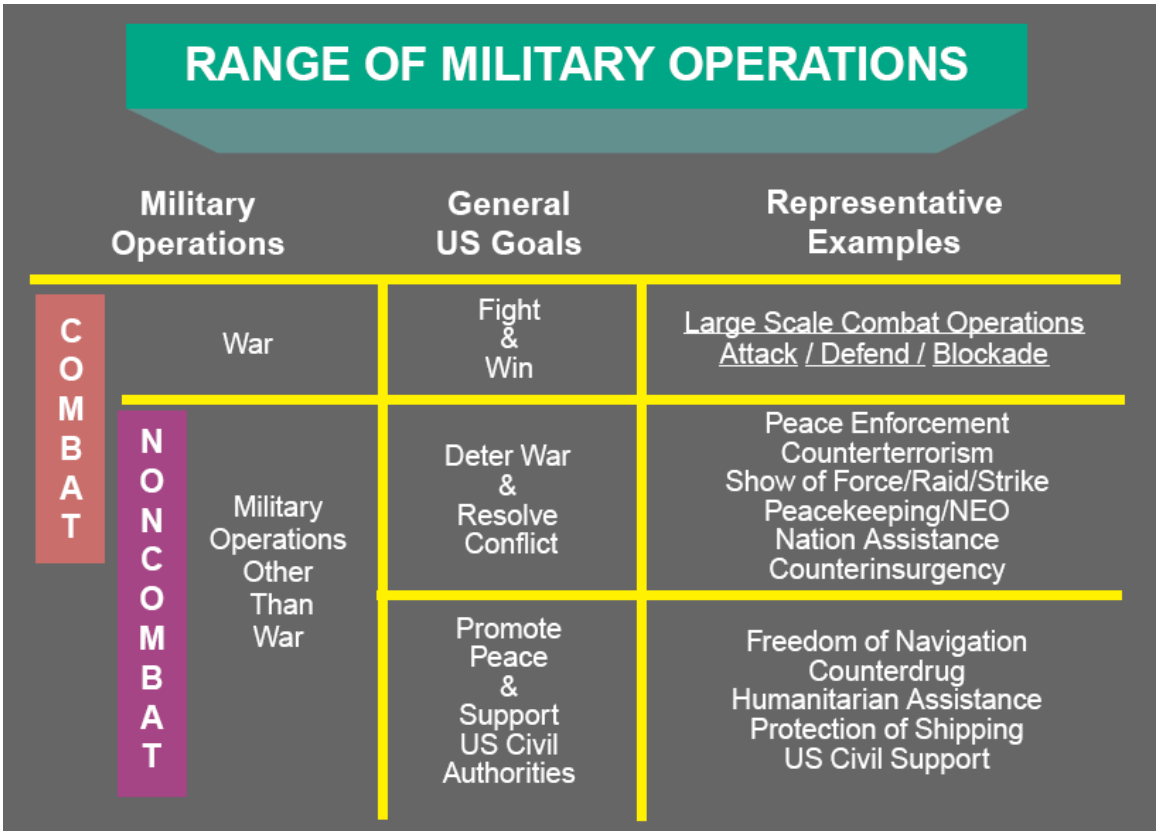


Figure 7: Range of Military Operations [5]

the introduction of predictive matrices. These matrices should enable the addition of different types of nodes, such as an intermediary port, without compromising the speed of the model.

The ease of adding types of connectors and nodes will be tested by incorporating Army assets as would be seen in Joint Logistics Over the Shore (JLOTS) , as seen in Figure 8. JLOTS exercises today include Navy and Army assets conducting cargo discharge operations, and include the interfacing of transportation modes in the surf zone, seaward of the surf line, and on the beach [13]. These operations include the transfer of Marine cargo between Navy and Army asset, as seen in Figure 9 as well as some unique capabilities. This includes the use of mobile piers to form austere ports, accessible by non-amphibious vessels [101]. The systems of JLOTS will serve as the foundation of the Sea Base, with the aim of overcoming the difficulties of current

JLOTS, such as the limited operations is increased sea state [81].

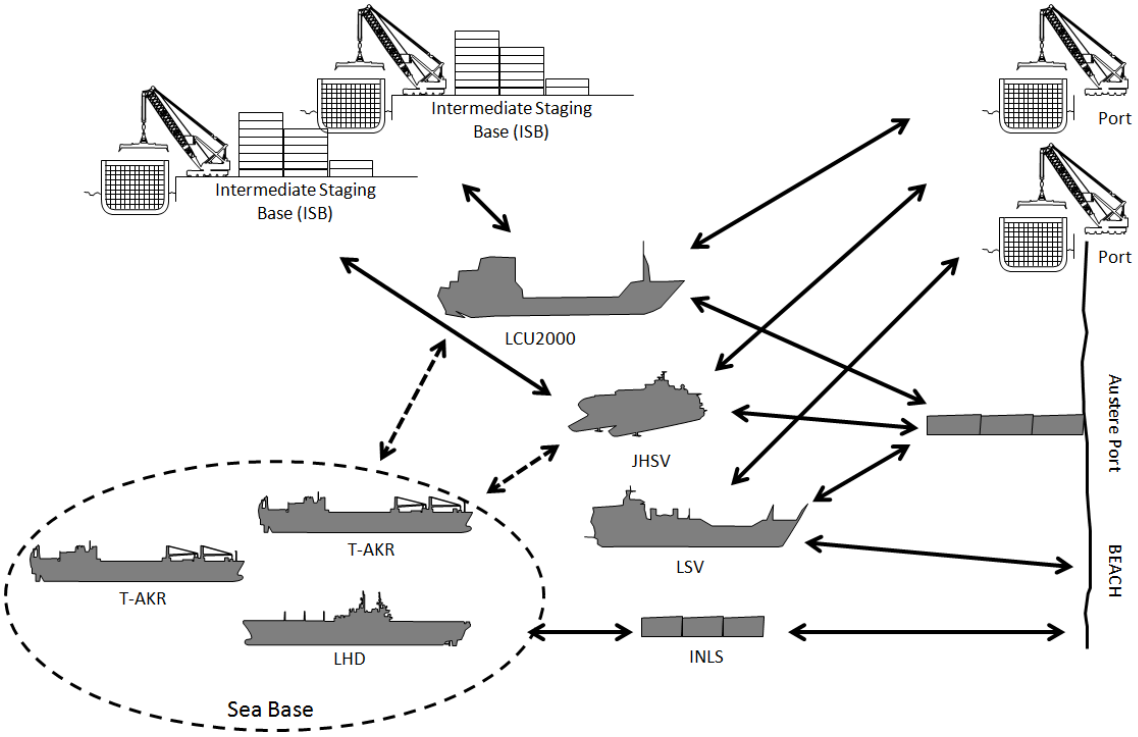


Figure 8: Additional Army Assets for JLOTS Assessment

5.2.1 Experiments

1. Runtime penalty for adding additional assets and cargo nodes

2. Validate and verify model results

5.2.2 Validation and Verification

It is impossible to validate a Sea Base model against real data because of the large scale of the operation modeled and the use of future assets. In the absence of physical data, the model can be validated by gaining feedback and buy-in from subject matter experts on the processes completed and the resulting trends. In addition, partial model results can be compared to accepted models. In this case, the processes and trends will be examined by research sponsors with the Office of Naval Research

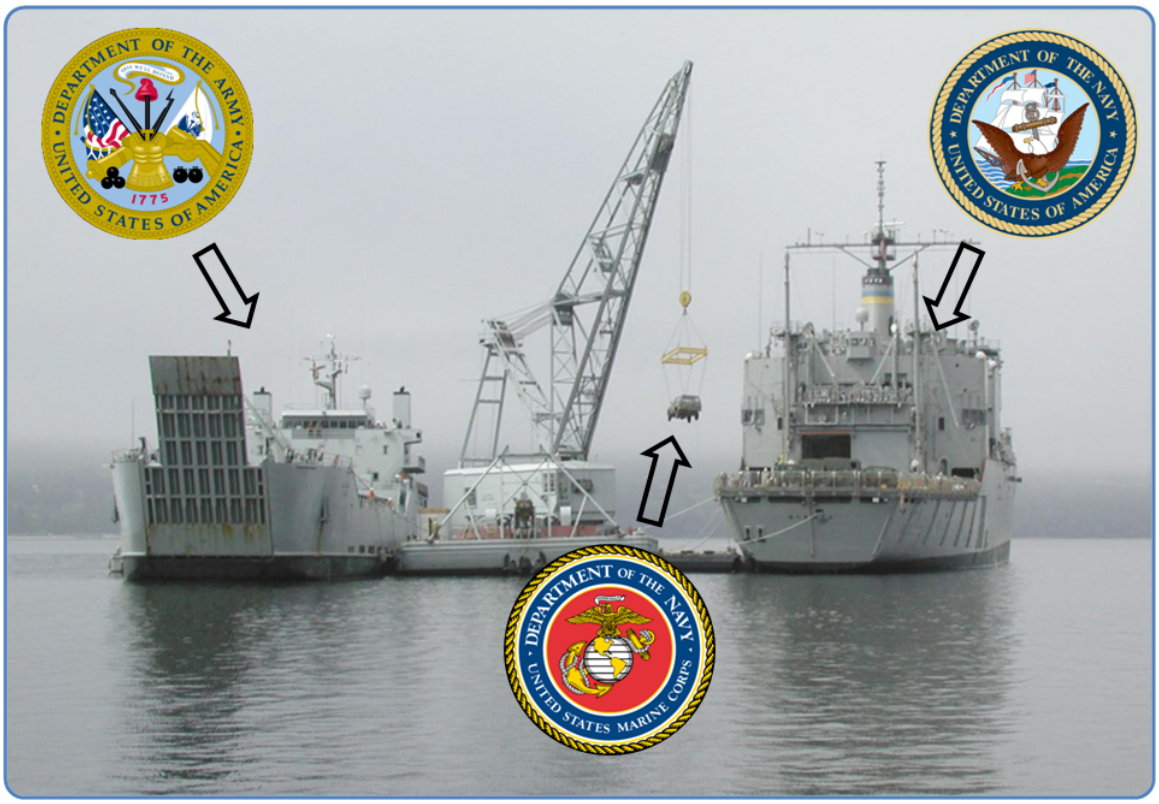


Figure 9: Participation in JLOTS [3]

(ONR). Specific cases will be documented and compared to released results from accepted models. The humanitarian scenarios generated by CDM [58] are well documented and can be recreated as specific cases to validate the results of this model. While this will only validate specific vessels, it will also provide an understanding that the method successfully represents processes. Additional vessels and assets will have to rely on the quality of data used to describe the process.

5.3 Dynamic Loading

A method was needed to load connectors dynamically based on the cargo needed and the size and lift capability. Two algorithms were suggested, a loading based on a prioritized list and using mathematical optimization and the knapsack was shown to be more robust. There are some shortcomings that will have to be researched

including the consideration of the location of cargo when selecting a load out.

5.3.1 Experiments

1. Compare algorithms such that selection of cargo minimizes introduction of additional wait time in delivery schedule

2. Document ability to match cargo desired and cargo delivered

5.4 Dynamic Routing

Dynamic routing allows decision making to occur at the connector level based on the current conditions of the operation. This results in the connector to be given options on loading and unloading locations. This will be necessary for the incorporation of intermediary ports and austere ports. For example, during the simulation, the MEC could have the option to choose between loading at the Sea Base, an intermediary port, or a primary supply location. To accomplish this decision, an accurate prediction of the queue time, loading/unloading time, and travel time. Since the load/unloading time is based on the interface used and the travel time is based on distances and vessel speed, the model must be able to predict the queuing time. The quality of this prediction is of interest.

5.4.1 Experiments

1. Level of effort to incorporate a change in possible locations

2. Quantify the trade-off between accuracy and run time

3. Compare accuracy of queue prediction, selection of cargo object and interface for different algorithms

5.5 Robust Design

Once the Sea Base model is developed and validated, it is necessary to facilitates real-time design in a cross scenario, system-of-systems analysis. This will be done using the robust design process described in Chapter 10. The performance measures of interest will be selected that are representative of military operations of different types. One possible measure of performance (MoP) is total time to complete the operation or shortfall, which is a function of the difference between cargo needed and cargo delivered over the time of the operation. The concept of shortfall is illustrated in Figure 10 where the total shortfall is the sum of the area between the shortfall and demand curves, seen in blue. The size of the time unit can be set to an interval of interest, for example, every day or every eight hours, and in the conceptual diagram is selected as every two time units. These MoPs will come with the associated goals,

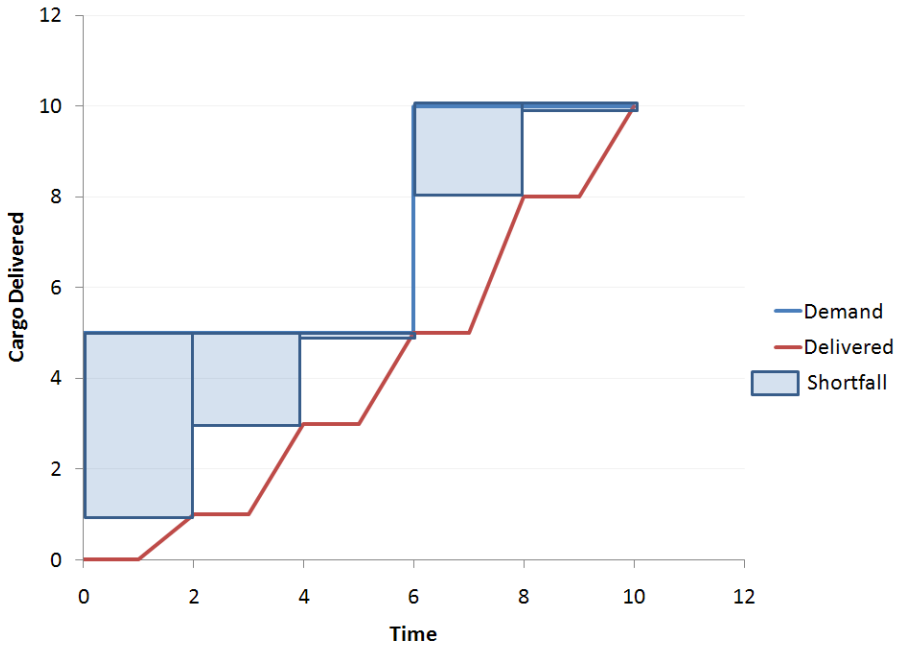


Figure 10: Concept of Shortfall

be it to match a target operation delivery time or to minimize the total shortfall. This goal will define the loss function of interest. The noise and design factors to be

considered will be selected with an initial possible list of factors is given in Table 1. The Design of Experiments (DoE) must be carefully selected to incorporate the use

Table 1: Potential Design and Noise Factors

Design Factors	Noise Factors
SES Speed	Cargo to deliver
Max Lift	Stand off distance
Interfaces available	Distances to supply bases
Time to load	Distances to intermediary ports
Time to beach and unload	Cargo on prepositioned ships

of feasible scenarios. A range of feasible scenarios is developed from the family of scenarios detailed in the previous section and the segmented design space is sampled. Once the design is selected, the model will be executed and appropriate metamodels developed to investigate the impact of the design and noise variables. This may require iteration on the variables selected and the loss function of interest. The results will be compared to identify the design drivers in individual scenarios and compare those to the design drivers across multiple scenarios.

5.5.1 Experiments

1. Identify measure of performance (MoPs) that are relevant to range of operations
2. Identify sampling methods for segmented spaces
3. Compare the feasibility of the sampling methods to complete coverage
4. Compare robustness results for complete coverage, and feasible scenario options
5. Identify the design drivers for the MEC

5.6 Research Objectives

The objective of this thesis is to formulate an approach to modeling a large scale operation, using the Sea Base as an operational example. This approach aims to

include the ability to capture the system-of-system interactions present, including the treatment of interfaces between vessels and other assets as design variables. The method formulation must be able to handle variations in the number and type of assets modeled. In addition, the method must be able to capture a variety of scenarios to represent the range of operations where Sea Base assets would deploy, detailed in Chapter 6. It is proposed that a hybrid method incorporating principles from matrix controllers and mathematical optimization into discrete event simulation fulfills these requirements, demonstrated in Chapter 7. By abstracting the loading, the cargo necessary cargo can be dynamically loaded to meet the demand with the connectors and cargo sources present, shown in Chapter 8. Chapter 9 works to make the supply chain a product of the connectors, cargo sources, and cargo available, with the introduction of dynamic routing. The Sea Base model allows for the design of a future Sea Base connector, such as the MEC, in the context of the future navy fleet. Through the principles of robust design across applicable scenarios, the design variables and choices that drive the performance are identified in Chapter 13 through 15. This thesis uses large scale system-of-system models in the design of future assets.

CHAPTER VI

SCENARIO DEFINITION

This chapter will present the reasoning behind the assertion that scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables. Example scenarios are developed to show their definition within the model framework to demonstrate the assertion requirements to:

1. Identify and quantify scenarios relevant to conceptual connector (MEC) design
2. Develop generic cargo vector including minimally transportable units

6.1 Use of Scenarios

Scenarios are a widespread technique to aid in decision making by considering uncertainty in the future [191]. These techniques have been used by military strategists throughout history but the modern application is attributed to Herman Kahn [45]. The goal of developing scenarios is not to try to predict the future but to stimulate discussion and insight into what the future could bring [155]. It is impossible to encompass all possible occurrences and the selected scenarios can always be faulted and found lacking [186]. But the use of scenarios enables the decomposition of complex phenomena into coherent, analyzable subsystems [167].

6.1.1 Military Application

Military planners have long dealt with uncertainty through the use of scenarios, looking at world futures as well as breaking it down to regional threats [93]. War gaming

has been used to explore these scenarios and these war games have spread from traditional combat to medical and cultural scenarios [127]. The variety and complexity of challenges the military faces is highlighted in the National Security Strategy [95] and was given in Figure 7, which showed the Range of Military Operations. The military can not simply plan for traditional operations but must balance those requirements with the capabilities needed for future operating concepts [10]. Operations have become more complex and can now include disarmament, demobilization, reintegration of combatants, community policing, diplomacy, conflict prevention, conflict resolution, and post conflict reconstruction [156]. Feist highlights the peril of ignoring either end of the spectrum, from conventional wars to nation building [75]. Scenarios are needed that balance the analysis of capabilities across traditional operations and broader range of potential operations.

6.2 Scenario Development

Before defining scenarios, it is necessary to detail the type of scenario to select and the information needed. Urwin, et al., require that a scenario satisfies the following criteria [189]:

1. Include multiple stakeholders requirements
2. Be applicable in multiple timeframes
3. Be sufficiently straightforward to be easily understood by non-experts, but at the same time sufficiently rich to be informative to domain and subject matter experts
4. Be plausible, in the sense of representing a possible future

Frankis, et al., work to identify military requirements for nonwarfighting operations, by using 'rule of thumb' expressions in the form of mathematical relations. These first-order approximations use situational factors, such as terrain and weather,

to select units and calculate the requirements [78]. Udeanu describes the necessary elements for a scenario including [187]:

1. The general theme of the exercise
2. The general objective
3. The performance period of the exercise
4. The exercises stages (sequences)
5. The themes of the exercises stages (sequences)
6. The stages (sequences) sub-objectives
7. The exercises managing and evaluation system
8. The subunit, unit, large unit or the commandment participating in the complex exercise, as well as real or hypothetical, support or enforcing forces and means
9. The hypothetical and real space for the exercises performance
10. Any kind of resource (hypothetical and real)
11. The hypothetical and real infrastructure
12. The operational context
13. The performance plan of the complex exercise

For this work, scenarios will be selected based on plausible future operations. These operations are defined based on the following characteristics:

1. Operation objective
2. Performance period
3. Participating units

4. Geographical layout - real or hypothetical

5. Performance plan

    - Including required resources

6.2.1 Scenario Selection

To capture all possible future scenarios would be impossible, so the scope is limited to possible applications of the system of interest, as was assumed by Thal and Heuck [179]. The goal of this work will not be to develop an exhaustive list of scenarios, but to present a small selection of plausible scenarios to demonstrate the models ability to capture the diverse aspects of the scenarios. The remainder of this chapter identifies and describes the scenarios that will be used for the robust design process. The selected scenarios are:

1. Large Scale Military Operation

2. Small Military Operation

3. Humanitarian Mission

4. Sustainment Operation

6.2.1.1 Large Scale Military Operation

The Marine Expeditionary Force is the primary Marine Corp organization, but the Marine Expeditionary Brigade (MEB) is a middleweight force that is light enough to use amphibious ships, but large enough to accomplish the mission [49, 50]. The MEB sized force, approximately 15,000 Marines, was reconstructed in 1999 [99] and is capable of responding to a large spectrum of conflicts, from humanitarian to warfighting - from assaulting an enemy beachhead to bringing ashore supplies to a hurricane-ravaged nation [76, 24]. Although traditionally the MEB phases ashore all of its

Table 2: Future Marine Expeditionary Brigade (MEB(F))

Category	Description	MEB(F)
Marine Platoon	36 Marines	78
EFV	Personnel Carried Internally	47
LAV-25	Personnel Carried Internally	27
M1A2	Personnel Carried Internally	47
EFSS Element	2 HMMWVs, EFSS, Ammo Trailer, 16 personnel	6
CEB Element	7 personnel, 1 modified M1A2	48
Arty Element	155mm HOW(T) and 11 personnel	21
HIMARS Element	HIMARS luncher and personnel	6

elements, the Sea Base allows the combat support, combat service support, and command elements to remain at sea [194]. Moving only the combat element ashore reduces the ship-to-shore movement requirements. The military is looking to be able to deliver two brigades as the assault echelon from amphibious ships [181]. The MEB would be delivered from 33 amphibious ships, 11 each of LHAs/LHDs, LSDs, and LPDs [140].

For this work, a large scale operation will be the ship-to-shore movement of two MEBs. Gen. James Amos laid down the goal of having a MEB be able to fit in 15 amphibious warships [73], forming the cargo delivery group. Based on figures in Strock's presentation [172], the MEB composition is given in Figure 11. The ground combat element will be delivered to shore and was decomposed to the minimal size that would be transported as a single unit. These cargo categories and the number to be transported for one MEB are given in Table 2 and are the minimally transportable cargo elements that will help form the generic cargo vector.

6.2.1.2 Small Military Operation

The Marine Expeditionary Unit Special Operations Capable (MEU(SOC)) is a specially trained unit to provide a capability to rapidly execute Amphibious Operation [94]. The ability to conduct ship-to-shore movement is part of the SOC qualification

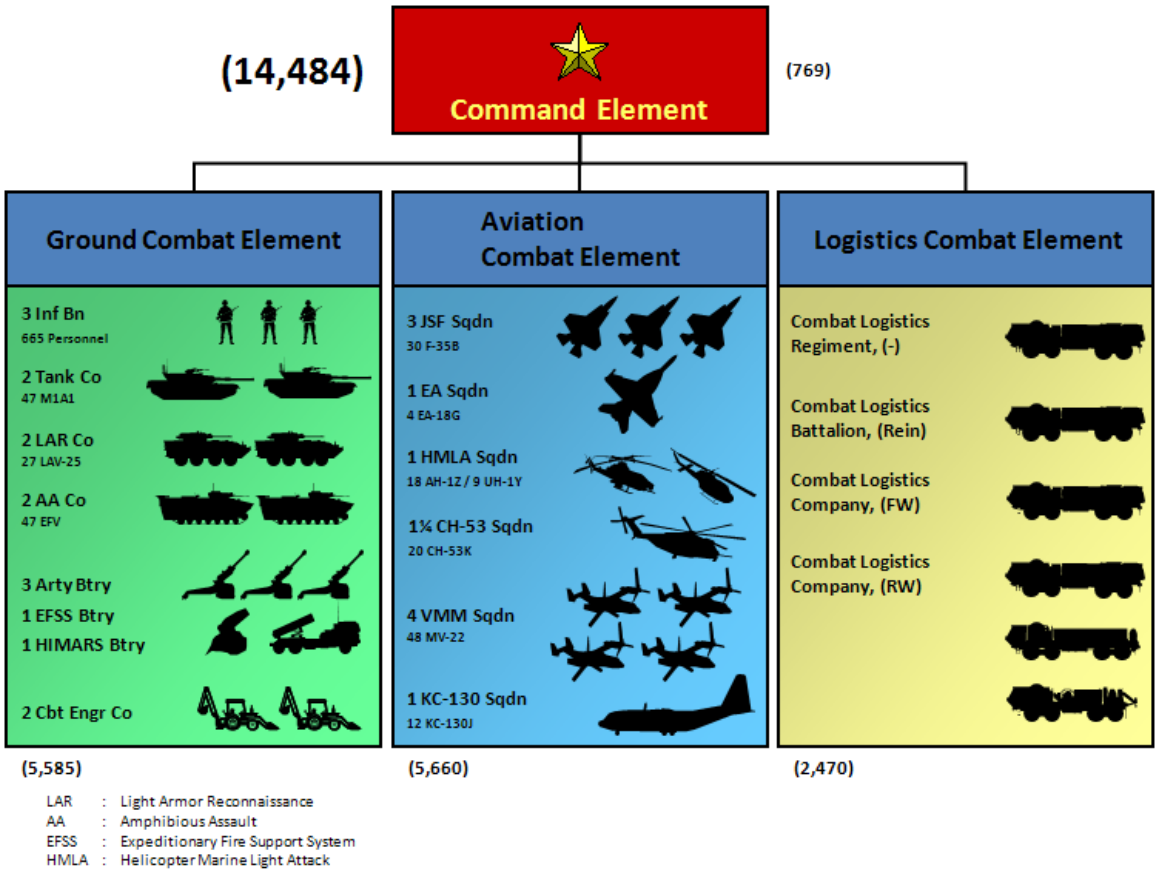


Figure 11: Future Marine Expeditionary Brigade (MEB(F))

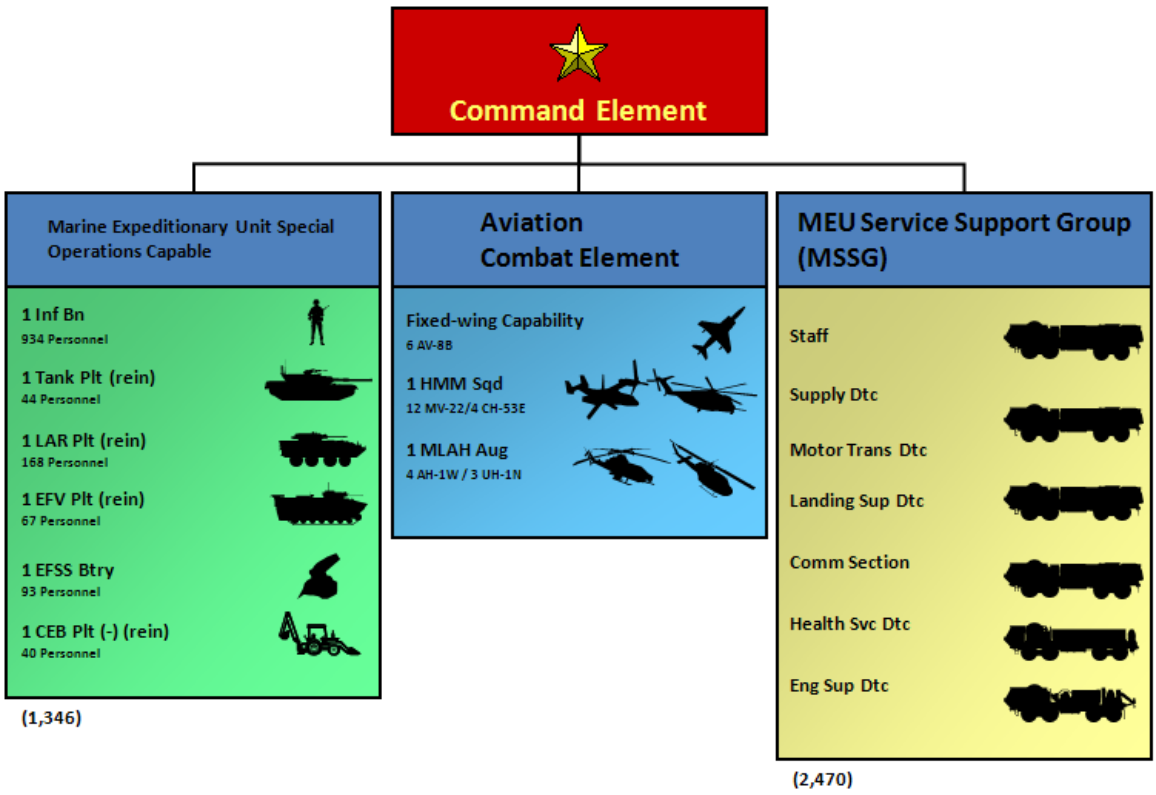


Figure 12: Marine Expeditionary Unit (MEU(SOC))

[110]. These units provide a sea-based, forward presence to respond to multiple mission types, including Amphibious Operations, Maritime Special Operations, Military Operations Other Than War, and Supporting Operations [7]. MEU (SOC) are a key forward presence and were used during the Gulf War [198], but are also prepared for other mission such as noncombatant evacuation operation (NEO) [173].

The small military operation scenario will be the delivery of a MEU(SOC) to shore from the three ships of an Amphibious Ready Group (ARG) [94]. Figure 12 shows the elements of a MEU (SOC) based on Hagan's thesis [91]. The ground combat element was decomposed to minimally transportable elements, listed in Table 3.

6.2.1.3 Humanitarian Scenario

It is important that the scenario definition process is broad enough to capture humanitarian scenarios. The Congressional Budget Office has identified the increased

Table 3: Marine Expeditionary Unit (MEU(SOC))

Category	Description	MEU(SOC)
Marine Platoon	36 Marines	26
EFV	Personnel Carried Internally	14
LAV-25	Personnel Carried Internally	4
HMMWV	Personnel Carried Internally	12
M1A2	Personnel Carried Internally	4
EFSS Element	2 HMMWVs, EFSS, Ammo Trailer, 16 personnel	6
CEB Element	7 personnel, 1 modified M1A2	6

frequency of operations other than war, including humanitarian and questions if the U.S. forces are prepared to meet the requirements of carrying out these missions since they require a different mix of forces and equipment from conventional warfare [6]. The United States Department of State states that in 2009, 335 natural disasters were reported causing 11,000 deaths, impacting the lives of more than 120 million people, and causing more than 41 billion dollars in economic damage around the world [20]. According to the Inter-Agency Standing Committee called for by the United Nations General Assembly to strengthen coordination of humanitarian assistance, military assets can be used when they provide a unique capability and when the operation is time limited [11]. Bessler identifies that in many cases humanitarian organizations have no option but to rely on military assets to deliver aid [40] and USAID acknowledges the criticality of the Department of Defense's transportation, logistics, and engineering capabilities in assisting in large scale disasters [20]. This criticality has been seen in disaster response, which is time sensitive and there are few options for delivery such as in Haiti where in January 2010, an earthquake killed more than 200,000 people [20]. A medical relief team from the International Medical Corps identified the importance of the military response not only in performing security but in providing tents, stretchers, medications, food, water, and other critical equipment and supplies [32].

Two example humanitarian scenarios were developed based on information provided by CDM Technologies, Inc, a continental United States (CONUS) and an international humanitarian aid situations. The CONUS scenario, based on a disaster occurring somewhere similar to the Gulf Coast, is represented by four population centers of 25 thousand refugees. The population centers are distributed along the coast and three Large Medium-Speed Roll-on/Roll-off (LMSR) ships serve as the source of the cargo. Initially two days of supply and shelter materials are delivered for the refugees at each location. Each refugee receives two MREs and 1.5 gallons of drinking water per day and it is assumed that ten percent of the population will require shelter. The follow-on day is to deliver one day of supply and construction materials, weighing a total of 9110 lbs per beach. The international scenario, representing a Haiti type situation, is based on five population centers of ten thousand refugees, supported by three LMSRs. The initial and follow on day requirements are the same but instead of MREs, refugees receive 460 g maize and 80 g beans per day and 25 percent of the population requires shelter.

M931 tractor trucks with M1076 trailers, each carrying an 8820 ISO container also called a twenty-foot equivalent unit (TEU), were assumed to be used in this analysis. It is assumed that a single container must carry one type of cargo, so the number of containers to supply the necessary cargo will be rounded up to the next complete container. The containers were assumed to be loaded with cases of water and MREs and 90 kg bags of maize or beans. The containers were loaded until cubed out by volume and the total cargo weight plus the tare weight of the trailer and container becomes the minimum loadable unit weight.

The connectors can be loaded with containers of different cargo, but are limited to a single type of containerization, in this case a TEU on M1076 trailers. The connectors are loaded based on the inputted cargo footprint area and lift capability, filled until adding the next container would exceed the weight or area limit. The

Table 4: Humanitarian Cargo Assumptions

Container Type	Domestic Day 1	Domestic Day 2	International Day 1	International Day 2
Water	19	10	8	4
MRE	9	5	0	0
Beans	0	0	1	1
Shelter	5	0	5	0
Construction	0	1	0	1

MEC will carry three prime movers and the SSC carries one, with the type of prime mover dependent on the type of container carried. The calculated weights, footprints, and number to be delivered are given in Table 4.

6.2.1.4 Sustainment Operation

The job of the Sea Base does not end with the delivery of forces, as the Navy and Marine cooperation extends to logistics [48]. The Navy plays a role in developing, deploying, employing, and sustaining the task force [47]. With the shift toward seabasing, the Marines will increasingly rely on support from the sea for fires, logistic, command and control and force projection [105].

The supply of troops is a challenge for the Navy, as was seen during Operations Enduring Freedom and Noble Eagle [27]. Operational Maneuver From the Sea (OMFTS) is especially challenging with the transportation of bulk fuel and water, currently supplied in 500 gallon pods [47]. There are many approaches to improving supply for OMFTS, including investigating civilian helicopters [80] and developing new vessels, such as the High Speed Vessel (HSV) [8]. The supply mission is common and a stresses the logistical chain, thus warrants incorporation as a distinct scenario.

The military provides a variety of resupply missions. The Military Sealift Command (MSC) transported 95 percent of the combat and military cargo needs of US warfighters [26]. In fiscal year 2009, this required the transportation of more than 4 million square feet of combat cargo and 2.5 billion gallons of petroleum products

[21]. Resupply is not limited to military operations as the military also resupplies the Antarctic research base with 84,000 square feet of food, household goods and research equipment and 5.5 million gallons of crucial diesel, gasoline and jet fuel [22].

Unlike the previous scenarios, resupply is a cyclic demand with a goal of meeting that demand with the least cost. The demand is based on the type of operation supplied but will always contain food, water, medical, and petroleum products. In a military resupply, replacement parts and ammunitions are also critical. The dry supplies could be transported in TEUs or on pallets. The petroleum would be transported in pods, such as the 500 gallon pod, assuming a commercially available bladder [2].

6.3 Necessary Information

The information that describes the scenario forms the inputs to the model. The geographical information defines the distances between locations, including the stand-off distance of assets and the distances to supply points. The units selected define the assets to incorporate including the number and types of ships, helicopters, and other vessels. The units of interest also play a role in the resources to be delivered. These resources are defined as a demand over time. This demand is listed in terms of a generic cargo vector. The generic cargo vector must encompass the demand for all scenarios.

6.3.1 Generic Cargo Vector

The generic cargo vector encompasses the cargo that would be necessary to deliver for any scenario. The Military Airlift Command in Airlift Deployment Analysis System classifies cargo types in terms of weight, volume, and square foot (foot print) measurement [152]. This information about the cargo will form the basic information that will be used to load vessels, as will be seen in the development of the loading algorithm in Chapter 8. The vector developed to encompass the scenarios in the

Table 5: Generic Cargo Vector

CargoType	Weight (LT)	Foot Print (sqft)
Marine Platoon	3.96	216
EFV	33.93	420
LAV - 25	12.59	172
HMMWV	2.63	106
M1A2	60.36	384
EFSS Element	9.43	400.7
CEB Element	61.14	426.2
Arty Element	7.03	112.8
Mortar Element	2.26	113.1
Antiarmor Element	5.54	212.4
HIMARS Element	10.71	182
Pallet	1.8	16
TEU - Water	17.08	160
TEU - MRE	9.53	160
TEU - beans	20.33	160
TEU - shelter	0.84	160
TEU - Eng Co	20.33	160
TEU - max size	21	160
Petroleum Pod (500 gal)	1.66	10

chapter, as well as having the potential to describe other scenarios is given in Table 5. The information used to develop these vectors was collected from many sources, government [131] and commercial, resulting in a minimally transportable unit, its gross weight in long tons and foot print aboard a transport.

CHAPTER VII

MATRIX FORMULATION

This chapter explores the hypothesis that introducing matrix manipulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduces the effort required to integrate new assets. A comparison will be performed for a traditional DES formulation and a matrix based formulation. Then the following experiments will be performed:

1. Runtime penalty for adding additional assets and cargo nodes
2. Validate and verify model results

7.1 Traditional vs Matrix Formulation

The model was initially formulated as a traditional DES using limited resources. The SimPy code for this implementation is given in Appendix A. The selection of loading spots was formulated as selecting the minimum time connection that had a spot available. The options for cargo transfer are using the starboard ramp (side), using the stern ramp (rear) or transferring cargo through an intermediary loading platform (through MLP):

$$\text{rear connection} = min(\text{number of open rear spots}, \frac{1}{\text{time to rear load}})$$

$$\text{side connection} = min(\text{number of open side spots}, \frac{1}{\text{time to side load}})$$

$$\text{using MLP connection} = min(\text{Can MEC load through MLP?}, \text{number of open MLP spots}, \frac{1}{\text{time to rear load}})$$

$$\text{connection to use} = max(\text{rear connection}, \text{side connection}, \text{using MLP connection})$$

The equations must be edited for each type of connection and any special conditions, such as the possibility of not being physically able to use an interface. The selected spot is then dealt with using a series of if-else statements to reserve that type of resource. Adding an interface option requires adding or modifying seven lines within the connector process and five additional lines throughout the code. This was unacceptable for building a flexible supply chain as the number of if-else statements to capture each case would be impractical.

The shift to matrix based considerations removes these if-else statements, and abstracts the selection of loading spot to use. This reduces the number of lines needed to define a connectors process. A majority of the code remains the same as the connector process steps have not changed, only the selection of the cargo interface. The results of these two methods were compared to check the processes abstracted to matrices. A basic example with six connectors, three cargo reloading spots each with two connection options, and two cargo unloading spots was run for 50 operational hours for each formulation. The investigation of matrix formulation was partially driven by the need to handle cargo better, so cargo was not considered here and the connectors simply choose the interface to use based on which is fastest. Figure 13 shows that both formulation capture an almost identical trend of unloading at the beach. This demonstrated that these methods perform the same for small problems so the selection is made based on ease of use.

7.1.1 Difference in Formulation

The incorporation of matrix based selection greatly reduces the number of lines of code required to define a vessel process. For a basic connector, this is a reduction from 31 to 18 lines of code. A description of the steps required in the processes follows. It is key to notice the matrix formulation uses the same process for loading and unloading so both segments do not need to be coded separately, as is done with

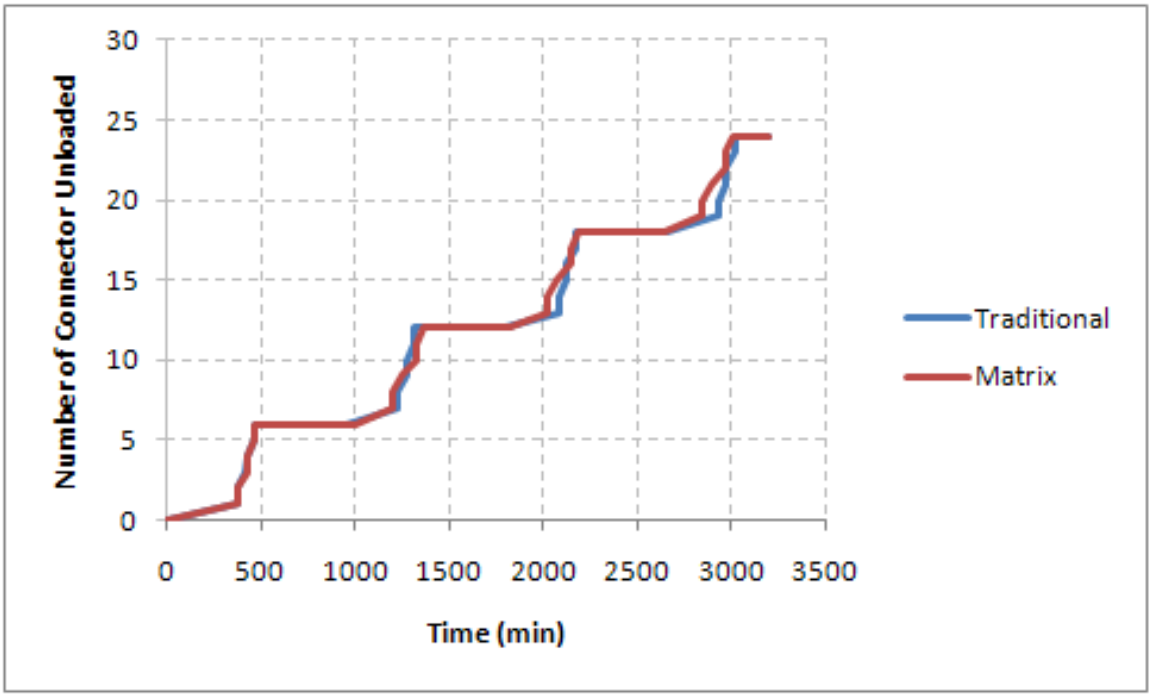


Figure 13: Traditional vs Matrix Resources

traditional resource pools.

Traditional formulation:

- Hold time to travel from Sea Base to transition distance
- Hold transition time
- Request shore spot from resource pool
- Wait time to travel to shore, beach, and unload
- Release shore resource
- Wait time to transition and return to Sea Base
- Request Sea Base spot from resource pool
- Identify load interface to use

  - Calculate minimum of spots in resource pool and inverse of travel time
  - Identify minimum of this minimum
- Request interface from resource pool based on calculations
- Hold for time to load, based on spot selected
- Release interface resource
- Repeat

Matrix Formulation:

- Hold time to travel and transition
- Identify cargo objects at location
- Identify interface spots on cargo supply ships with desired cargo
- Identify shortest interface and its associated wait time
- Hold wait time
- Release interface spot
- Change mission
- Repeat

The incorporation of matrix formulation does incur a time penalty, as seen in Table 6. This penalty was assessed for 3 LMSRs and varying number of connectors. The matrix formulation is moderately more complex as it includes the cargo requested to be loading in selecting a LMSR to use. The difference in run time may seem large but is still less than a third of a second so this penalty is acceptable. The run time will be further explored as the code complexity is increased.

Table 6: Run Time Comparison - Traditional vs Matrix

Connectors	Traditional (sec)	Matrix (sec)
6	0.005	0.049
12	0.011	0.164
18	0.012	0.257
24	0.015	0.308

7.2 Use of SimPy

SimPy is an object oriented, open source package originally developed by Klaus G. Müller [132]. It solves the platform dependency issue of older process-oriented software by developing a package within Python [124]. Python is a platform independent, open source scripting framework and a wide variety of Python-based projects are available [85], which take advantage of Python's elegant language and ability to perform complex operations [34]. The use of SimPy will enable the incorporation of other Python modules as needed for sub-processes within the model. This thesis uses SimPy 2.2, which is available along with examples on the SimPy website [4].

SimPy offers the additional advantage of allowing modular development [57]. This enables the abstraction of the subprocesses within the model and the reuse of common code, such as the spot selection algorithm described previously. The use of Python classes allows for vessels and vehicles with similar properties to share the definition of their process. This reduced with coding difficulty to add new vessels that share the same type of process as an existing asset.

7.3 Penalty for Expansion

The penalty for expansion of the model is tested by adding Army assets that would be seen in Joint Logistics Over the Shore (JLOTS) operations, as was discussed in 5.2 and shown in Figure 8. The Army assets include logistics support vessels, landing craft, and lighterage [121]. Some of the vessels fit into existing classes, connector in the case of the LCU 200, JHSV, and LSV. The INLS is carried into theater, but self

deploys from the Sea Base. An austere port represents an additional landing option at the beach and will have the same instantiation as beach heads. The code penalty will be discussed for the two options, where the process exactly matches an existing vessel and where a process must be modified.

7.3.1 Vessel Matching Existing Class

If a new vessels process exactly matches an existing class, it is very simple to add the new vessel type. The key attributes of that type of vessel are defined as inputs and added to the input file. Within the model, the vessel type must be initialized and the class defined, as included below. The example given here is for the LSV which serves as a surface connector, which is the defined category for the Supply Ship class.

```
class LSV(SupplyShip):
    list = []
    def __init__(self, name, spots, wait, max_lift, lift_eff,\
            max_area, area_eff, MTBF, MTTR, compatibility, speed,\
            push_locs, pull_locs, fuel_usage):
        SupplyShip.__init__(self, name, spots, wait, max_lift, lift_eff,\
            max_area, area_eff, MTBF, MTTR, compatibility, speed,\
            push_locs, pull_locs, fuel_usage)
        LSV.list.append(self)
        self.deploy(LSV.list)
```

7.3.2 Vessel Requiring New Class

The INLS is carried into theater and then self deployable from the Sea Base, which is not an existing class. Once the INLS is assembled at the Sea Base, it follows the same process of a surface connector, so it is possible to create a new process or create an option within the existing processes. Since the only difference is the need to be carried into theater, a class was created for an organic connector and the flag below

added to the general connector process. This subclass is initiated when the INLS is called, creating an organic connector. The information to define this vessel is the same as the general connector and these values must be added to the input file.

```
if isinstance(self, Organic):
      yield put, self, store, [self]
      yield waitevent, self, self.SBSignal
```

The INLS must be carried within another vessel, so its deployment modifies the process for that vessel as well. As it will be carried aboard one or more of the Sea Base cargo ships, the general process for the Sea Base cargo ship was modified. An input was included for the maximum number of INLS that can be carried. When the cargo ship is first instantiated, it determines the number of organic assets that will be carried aboard and a check for the INLS was added, as seen below. Once the cargo ship reaches the Sea Base, the INLS are reactivated using a SimPy event, and proceeds as a general connector. A SimPy Event is used to maintain the properties and status of the vessels carried on the cargo ship and this construct is used throughout the vessel processes when one vessel is transported by another.

Before departure:

```
for r in range(self.Num\_carriedINLS):
      yield get, self, ISBINLS, 1
      self.INLSid.append(self.got[0])
```

After arrival:

```
 for s in range(self.Num\_carriedINLS):
       whichINLS = self.INLSid.pop()
       whichINLS.SBSignal.signal()
       yield hold, self, INLSTimeToOffload
```

7.3.3 Run Time Penalty

Since classes are only activated with the presence of that vessel in the simulation, the addition of types of vessels does not impact the run time of the model. Instead, the speed of the model is determined by the number of vessels and possible nodes. The run time for additional nodes is a consideration of the routing algorithm, so only the number of vessels will be considered here.

A set of basic cases was established to test the relative run time. This case will be a single type of Sea Base ship and connector, in this case the LMSR and MEC will be used. The demand will remain the same, at 3000 pallets. The unloading spots and cargo available at the Sea Base will be great enough to not impact the simulation results, so all simulations will carry out the same number of trips to shore. The number of LMSRs and MEC will be varied and the total run time tracked in Table 7. The run time increases as the number of vessels increases with the number of vessels. For the same number of vessels, connectors add more time than supply ships, which can be attributed to the greater complexity of the connector algorithm. The connector calls the algorithms to select cargo at the Sea Base and to select the interface spot to use at the Sea Base and the Beach.

Table 7: Run Time Comparison

Number LMSR	Number MEC	Run Time (sec)	Per Vessel (sec/vessel)
1	1	4.65	2.32
1	10	8.10	0.74
10	1	5.02	0.46
10	10	5.80	0.29
10	100	16.65	0.15
100	10	10.28	0.09
100	100	22.74	0.11

Since the connector is the more complex type of vessel, the trend of run time was further analyzed. The cases were repeated for 3 LMSRs and increasing number of connectors. Figure 14 shows the growth for 1 to 100 connectors. The growth is

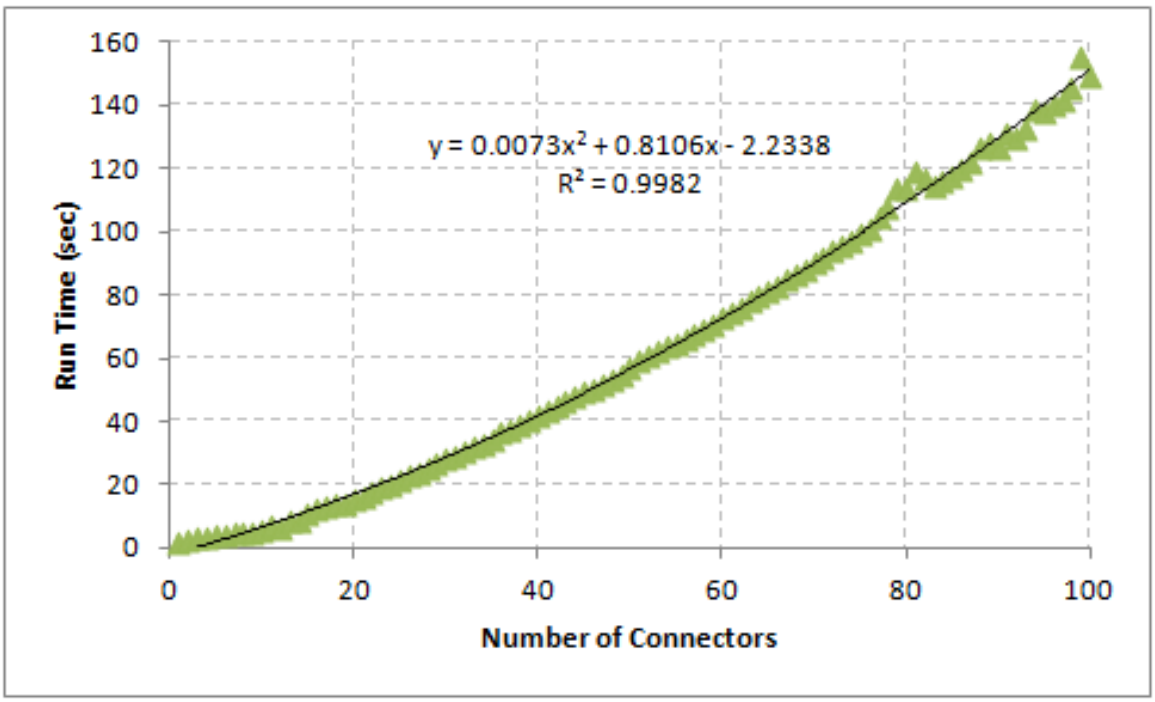


Figure 14: Run Time per Connector

slightly greater than linear, with a quadratic term of 0.0073, but the trend is very close to 0.8 sec increase in the run time per connector added. The growth in run time is acceptable and demonstrates the expandability of this DES formulation. The run time of the model increases with the number of vessels incorporated and the relative complexity of their processes.

7.4 Validation and Verification

Comparison of this formulation, as well as the dynamic loading, to the results of an existing model will be discussed in Chapter 11. Verification of the behavior of the model was performed by having Naval experts give insight into the processes used in a logistics operation. The vessel processes are described in Chapter 10. The resulting trends and interactions were reviewed by experts courtesy of ONR.

CHAPTER VIII

DYNAMIC LOADING

The hypothesis that knapsack loading is an efficient and robust approach for solving the loading sub-problem, was developed in section 4.3. The following experiments will be run and discussed.

1. Compare algorithms such that selection of cargo minimizes introduction of additional wait time in delivery schedule

2. Document ability to match cargo desired and cargo delivered

8.1 Algorithms to Address Loading

As was discussed in Section 4.3, a knapsack type formulation is theorized to be able to solve the loading problem. The knapsack formulation and prioritized loading algorithms will be implemented and compared. The implementation of the formulations are given in the following sections.

8.1.1 Prioritized Loading

Prioritized loading is based on the user providing a prioritized list of cargo to be delivered. This prioritized list is a full list compiled in order of desired delivery to the shore. An example mathematical formulation follows:
Identify cargo that needs to be brought to shore and will fit on connector. The

connector is looking to load cargo so its mission is to pull cargo from cargo objects

$$\text{Cargo Schedule: } \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}$$

Cargo Schedule should already be prioritized by order to arrive to shore Weight vector $[\,10\ 15\ 20\,]$ (in LT)

Area vector $[\,50\ 100\ 120\,]$ (in sqft)

Compatibility Vector of $[\,1\ 1\ 1\,]$, so the connector can carry any type of cargo that will fit aboard. If any type is not compatible, the value in the weight of that cargo type becomes 10^6.

$$\text{Form Weight and Area vectors: } \begin{matrix} \text{Weight} \\ \begin{bmatrix} 10 \\ 10 \\ 15 \\ 15 \\ 20 \\ 20 \end{bmatrix} \end{matrix} \quad \begin{matrix} \text{Area} \\ \begin{bmatrix} 50 \\ 50 \\ 100 \\ 100 \\ 120 \\ 120 \end{bmatrix} \end{matrix}$$

Calculate cumulative sums and identify where the maximum weight and area are exceeded. For this example, maximum weight is 60 LT and Maximum area is 500

sqft.

$$\begin{matrix} \text{Weight} \\ \begin{bmatrix} 10 \\ 20 \\ 35 \\ 50 \\ 70 \\ 90 \end{bmatrix} \end{matrix} \quad \begin{matrix} \text{Area} \\ \begin{bmatrix} 50 \\ 100 \\ 200 \\ 300 \\ 420 \\ 540 \end{bmatrix} \end{matrix}$$

The first four units of cargo can be carried on this trip. The cargo want vector becomes $[2\ 2\ 0]$, a desire to load two of type 0 and two of type 1. There will be 50 LT and 300 sqft of cargo space used during the trip. These values are tracked to determine the average lift and area used.

8.1.2 Mixed Integer Linear Program (MILP)

The knapsack formulation takes the form of a MILP. If all of the cargo categories are categorical, that it is in Integer Program, but the general form is maintained. Using an MILP does not exclude continual variables, such as bulk water and fuel in holds. The user inputs a total demand for each cargo category and the priority of delivering that type of cargo. The formulation of this problem is:

$$Maximize \sum_i P_i x_i$$

subject to:

$$\sum_i weight_i x_i \leq maxlift$$

$$\sum_i area_i x_i \leq maxarea$$

$x_i \geq 0$ for all i

$x_i \leq needed_i$ for all i

Where $weight_i$ is weight of one item of category i, $area_i$ is area needed to transport one item of category i, and $needed_i$ is demand for that cargo category.

Prioritization as the value vector does not work directly because prioritization is defined as one being the most important, so maximizing

$$\sum_i P_i x_i$$

would cause the lower priority to be taken first. Using the inverse of the prioritization vector as the value of carrying that cargo leads to a bias toward smaller and lighter cargo items. To unbias this, the value function was normalized for priority by weight and area. The priority can be interpretted as a relative priority per unit weight and area. A zero priority is not possible, so one is added to the numerator to prevent an unsolvable linear programming problem.

New Priority (P_i):

$$\frac{\frac{Cargoweight_i}{Avgweight} + \frac{Cargoarea_i}{Avgarea} + 1}{Priority_i}$$

8.1.2.1 Implementation of MILP

It is necessary to solve the MILP each time the cargo selection algorithm is run so it must be incorporated into the discrete event simulation. Linderoth and Ralph discuss several open source solvers and LPSolve was identified as a branch and bound solver that serves as a callable library [119]. This library can be interfaced with a large number of languages, including Python. Version 5.5 will be used for this thesis and is available online [37].

8.2 Prioritized Loading vs Knapsack

Prioritized loading is the first formulation that incorporates loading of cargo onto the connectors. This cargo formulation is described in 4.3 where a prioritized list of cargo forms an input to the model. The goal here will be to deliver 3000 pallets to shore as soon as possible. If each cargo ship is assumed to carry 1500 pallets each, once

all the pallets aboard the two ships are delivered, no more cargo is available. The shortcoming of this method will be demonstrated by adding one large piece of cargo, which exceeds the lift capability of the connector, after 1000 pallets. This causes the delivery of cargo to stop once this item is reached because the cargo selection can not progress until this item is brought to shore. This problem was solved by treating the loading as a knapsack problem. The time history results of the cargo unloaded at the final destination calculated in these experiments are depicted in Figure 15 where the prioritized, knapsack, and knapsack with large item added align, so the knapsack does not suffer the same problem as the prioritized.

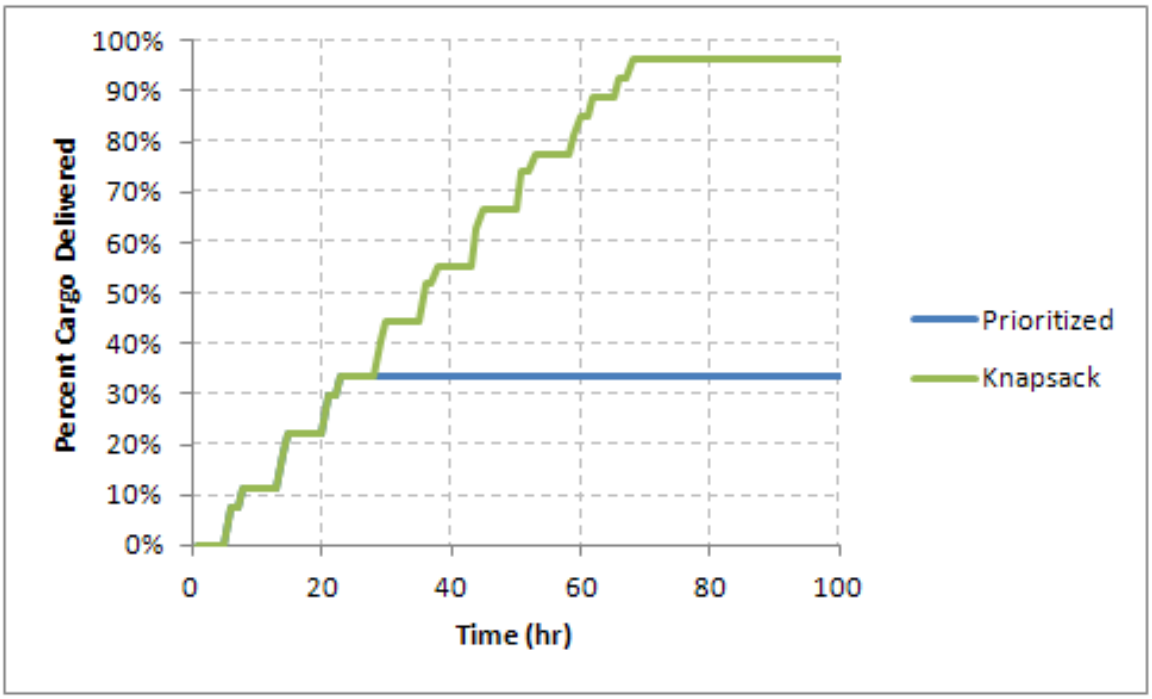


Figure 15: Prioritized vs Knapsack Loading

8.3 Shortcoming of MILP

The knapsack algorithm, as well as the prioritized loading, have a problem with the locational distribution of cargo. In the example above, the amount available was exactly the demand but the plots in Figure 15 show that the percent of cargo

delivered fails to reach 100 percent. This occurs when the demand for cargo is larger than the amount of cargo at one location. In this case, each connector can carry 111 pallets, and when the simulation terminates, each cargo source has 57 pallets remaining, matching the 114 needed on shore. There is no source that can provide all 111, so the connectors stop. If there is extra cargo available, the delivered cargo reaches 100 percent, seen in Figure 16, but it is not realistic to expect to have extra cargo. This problem expands as mixes of cargo are needed, where the cargo is needed and can fit on one connector but is not located at or on one cargo source. The knapsack algorithm needs to be enhanced to incorporate the location of the cargo when selecting the load out.

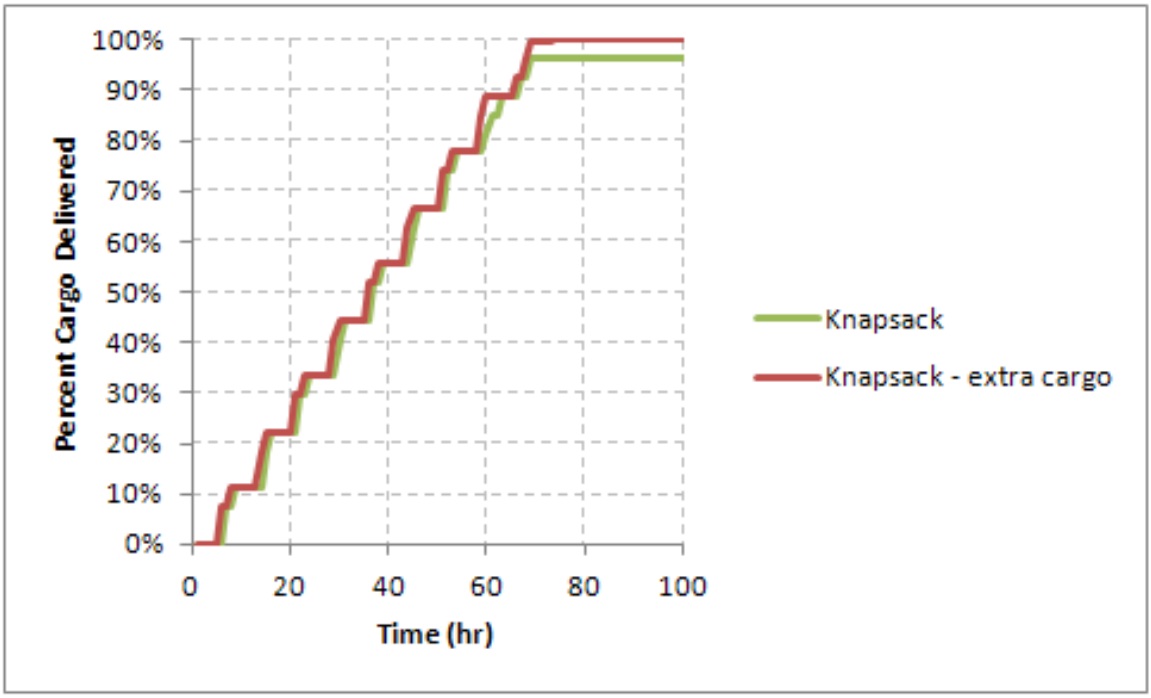


Figure 16: Knapsack With and Without Extra Cargo

8.4 Incorporating Cargo Locations

The existing MILP formulation does not account for the cargo available at cargo supply nodes. It is thus possible to have the simulation stall because the prioritized

load is not available at all. The selection of cargo is done separately with the amount of cargo available only being considered when selecting the load point to use. If the desired cargo is not available at any point, all the connectors will continue to attempt to load the same cargo without being able to move forward. The solution to the simulation stalling issue is to incorporate the cargo available at each loading point within the cargo selection algorithm. Thus it is known that the cargo combination exists on or at some load point. This will keep the simulation from stalling as was seen in Figure 16.

In essence, this is adding an assignment problem to the knapsack formulation. An assignment problem is selecting exactly one person to do each job [169, 134], or in this case, one job is assigned. Assignment problems are solvable for as many as 500 binary variables [137], so this optimization should be solvable for any realistically sized problem, especially since it is a simplified assignment of only one task. Assignment problems can be easily combined with other constraints, such as overtime as performed by Nauss [136]. Incorporating an assignment problem adds constraints for each cargo load option, of the form given below. These variables will be added to the cargo category decision variables, so the decision variables are divided into cargo to load (x_i) and load point to use (y_j).

$$y_j = \begin{cases} 0 & \text{if not using load point j} \\ 1 & \text{if using load point j} \end{cases}$$

8.4.1 Formulation of MILP

Adding the assignment problem constraints to the knapsack constraints, creates the following formulation:

$$Maximize \sum_i P_i x_i$$

subject to:

$$\sum_i weight_i x_i \leq maxlift$$
$$\sum_i area_i x_i \leq maxarea$$
$$x_i - \sum_j cargo_{j,i} y_j \leq 0 \text{ for all i}$$
$$\sum_j y_j = 1$$
$$x_i \geq 0 \text{ for all i}$$
$$x_i \leq needed_i \text{ for all i}$$

Where $cargo_{i,j}$ is the amount of cargo category i available on or at load point j. The former constraints remain the same with $weight_i$ as weight of one item of category i, $area_i$ as area needed to transport one item of category i, and $needed_i$ as demand for that cargo category.

This formulation is compared to the basic knapsack formulation in Figure 17. The simulation no longer stalls when the cargo selected is the remaining demand, but the available cargo is distributed between multiple load points. The algorithm can now match the available cargo and complete the delivery.

The additional constraints makes the MILP a more difficult problem to find an optimal solution. To test for additional computational demand, the cases run in Section 7.3.3 were repeated for the two formulations of the MILP, seen in 8. The run time is slightly greater for the MILP with additional location constraints, except for the first case. The increase in run time is not of large significance.

8.5 Selection of Loading Algorithm

The experiment addressed in this chapter are to select the loading algorithm based on matching the cargo demand and not introducing additional wait time. This wait time was seen in this example as the stalling of the simulation and failing to complete

Table 8: Run Time Comparison - MILP

Number of Vessels		Run Time (sec)	
LMSR	MEC	MILP	With Location
1	1	4.75	4.65
1	10	6.47	8.10
10	1	4.90	5.02
10	10	5.54	5.80
10	100	13.90	16.65
100	10	9.57	10.28
100	100	20.21	22.74

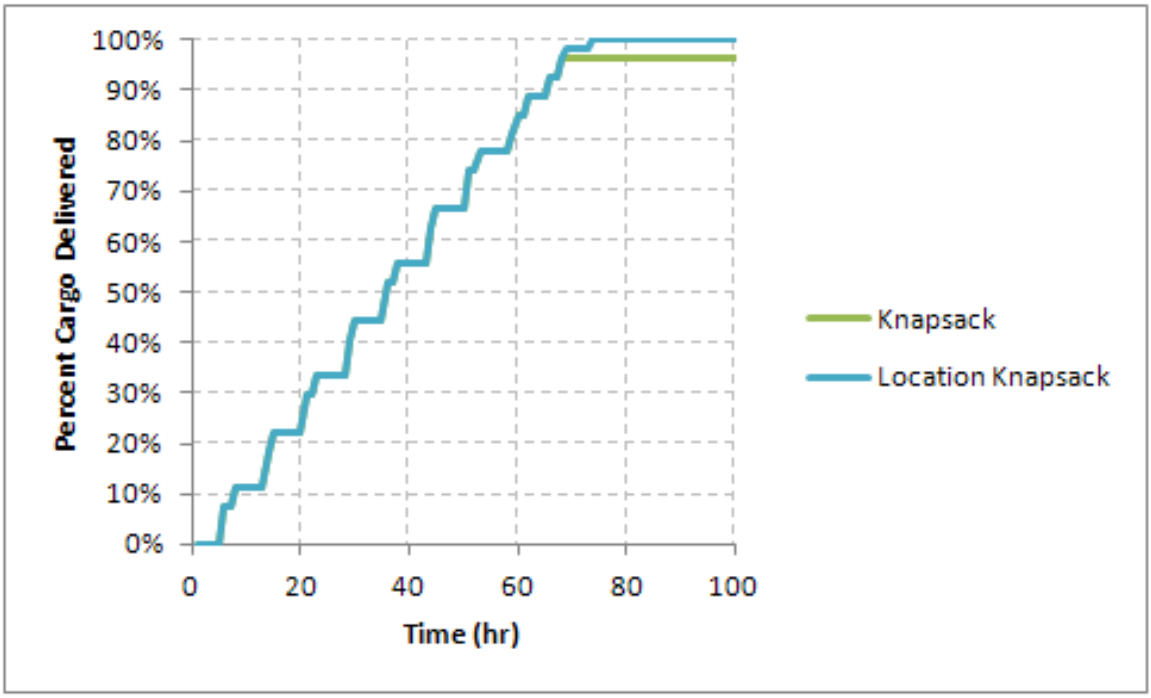


Figure 17: Knapsack with Additional Constraints

the delivery. Combining an assignment problem with the knapsack loading yields an algorithm that can overcome this stalling and match the delivery of cargo. The additional constraints do not cause the MILP solver, LPSolve55 in this case, to slow. This combination algorithm is an efficient and robust MILP formulation for the solving of the loading problem.

CHAPTER IX

DYNAMIC ROUTING

The need for and challenges of dynamic routing were detailed in section 4.4, with the hypothesis that matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing. To show this hypothesis, the following experiments were selected:

1. Quantify the trade-off between accuracy and run time
2. Compare accuracy of queue prediction, selection of cargo object and interface for different algorithms
3. Robustness of algorithm to disturbances

This chapter will detail existing methods of routing vehicles within a simulation, deriving concepts from traditional vehicle routing and computer science applications. These methods will be distilled to concepts that can be applied for this thesis. These concepts will then be tested using the model construct.

9.1 Dynamic Routing in Traditional Vehicle Routing

Traditional vehicle routing problems have been expanded to incorporate stochastic optimization techniques to optimize the selection of routes and loading points. According to Arellano-Garcia and Wozny, the stochastic properties due to consideration of uncertainties/disturbances is necessary [29] since dynamic demand can introduce queuing phenomena [145].

The need to break down the problem into multiple stages has been called reoptimization in many sources [107, 70, 39, 69, 168]. In reoptimization, routing decisions

are based on the current system state, which is updated upon arrival at a location [138]. This reoptimization is considered in various degree with the most basic consideration involving the original paths being followed unless a route fails and the most complex with reoptimization at every location [70]. Reoptimization is a better option when the demand is known at the start of the route [39] and it is expected to reduce average cost but at the expense of computational power [35].

The routing decision can be formulated as a Markov decision process model (MDPM), but the solution is difficult due to the large size of the state space [69]. Several ways to solve this type of model are investigated in the literature including approximation algorithms [38]. One approximation, chance constrained models, works to minimize expected cost based on decision variables using bounded penalty models [108, 151]. The chance constrained optimization problem has the following general form [113]:

$$minf(x,u,\xi), s.t. g(\dot{x},x,u,\xi) = 0, x(t_0) = x_0, h(\dot{x},x,u,\xi) \geq 0$$

where f is the objective function, in this case to minimize total trip time. The vectors g and h represent the equality (model equations) and inequality constraints. x, u and ξ are the vectors of state, control and uncertain variables. These problems are traditionally relaxed and solved as a non-linear program. This approach has been applied to traditional vehicle routing problems with uncertainty in travel time between locations and was solved using a genetic algorithm [82].

9.1.1 Critical Routing Considerations

By investigating algorithms and techniques used in traditional vehicle routing, key concerns about dynamic routing have emerged. The goal of any dynamic routing algorithm is to minimize the expected cost of a route [138]. Rollout algorithms have used simulation and function approximation to estimate the expected cost [168]. Dynamic routing to account for stochastic demand, in order to develop a robust algorithm, have

been incorporated into a discrete model by Pavone, et. al., with the system state defined by the departure times and vehicle loads where demand requests are seen as disturbances [145]. This work highlights the importance of any routing algorithm being robust to system changes.

Psaraftis raises an important question on how close to optimal a routing algorithm can be based on information only currently available rather than including future information [151]. If the demand in the problem is not stochastic, this future information is available. Technology has introduced benefits and complication to real-time routing as the amount of real time information available has greatly increased as well as the ability to predict information such as traffic condition [115].

9.2 Routing from Computer Science Perspective

As was discussed in Chapter 2.1, the problem considered in this work does not map directly to a traditional vehicle routing problem thus routing was examined from a computer science perspective. In the field of computer science, real-time routing is necessary for call centers, routing of packets in wireless internet, and network-on-chip systems. The development of routing algorithms for these fields has lead to many recent publications and provides insight into key components of a dynamic routing algorithm.

9.2.1 Call Centers

The objective of a call center is to minimize queue length and the waiting time spent by customers. In a heterogeneous call center, where servers do not service customers at the same rate, it has been proven that queuing the faster servers first can give better results even if a slower server is open [30]. Lin proved that for two servers with a common queue there exists a computable threshold above which customers should be routed to the slower server [117]. This work was expanded for multiple servers and a computable threshold was proven [120, 157]. Luh and Viniotis went

on to prove that for a system with N heterogeneous servers, threshold routing is the optimal customer allocation policy [120].

The literature reviewed has all related to input queued centers, where there is a common queue, but the model in this work has geographically distributed queues, so a common queue is not applicable. Stolyar expanded the threshold method for output-queued version [171]. In an output-queued model, each customer is assigned to a server upon arrival, also called immediate routing. Stolyar recommends queue assignment based on estimated unfinished work or using estimated average values.

9.2.2 Packet Routing

Packet delays for internet protocols consist of a propagation delay, transmission delay, slot synchronization delay and queuing delay, all of which must be considered when designing a routing algorithm [163]. Existing routers are often based on shortest path preferred algorithms, but work has been done to balance loads by incorporating a link utilization cost into the path selection [72]. Traffic efficiency was also increased by delivering along the path the minimum product of sum degree of nodes and neighbor nodes queue length [201].

Information packets are also transfered in wireless sensor networks. Previous control algorithms throttled incoming traffic to limit packet loss, but Ren, et. al., suggests a traffic-aware algorithm. Their work suggests developing a virtual potential field using a combination of a shortest path field and a queue length field to select the path [153].

9.2.3 Network-on-Chip

Dynamic, also called adaptive, routing is necessary in network-on-ship systems to meet the dead-lock free and real-time optimization decision making requirements. Mak, et. al. [122], suggests methods for routing algorithms in this network to build on previous approaches that exploit only local traffic information. To further improve

traffic balancing, the number of packets in a buffer is included in the calculation of the shortest path. To reduce computational requirements, the algorithm looks k steps ahead in the routing between nodes and can effectively avoid hot spots or faulty components. This avoidance increases the reliability, one of the key objectives in on-chip networks [190].

9.3 Routing Algorithms

Based on the computer science and vehicle routing considerations, two objectives were selected to measure the performance of a dynamic routing algorithm:

1. Minimize the wait time
2. Robustness to network disturbance

For this model, the wait time can be measured in terms of the time spent by each vehicle in the queues, waiting to load and unload cargo. Of more concern to the scenario is how much wait time is introduced in delivering to the final location. This will be captured in comparing the cargo arrival curves at the final destination. The robustness to network performance is key for this model as the logistical train would need to continue if a vessel is disabled or destroyed. The impact of a disturbance will be measured by its impact on the time to deliver cargo to shore.

Concepts and algorithms introduced in the literature discussed are applied and categorized in to the follow concepts:

1. No dynamic routing
2. Current states
3. Predicted states
4. Predicted states with retesting
5. Predicted states with current state corrections

These concept will be further detailed and then experiments performed to measure the impact on wait time and robustness to disturbance.

9.3.1 No Dynamic Routing

In this case, the connectors will have predetermined nodes to connect and will only travel between a single set of nodes. The connectors will simply travel from one location to an unload point and back, repeating the same process until no cargo remains to be delivered or the cargo needed does not exist at the load location. Only the location of the node, not the exact interface is selected. The interface is selected upon arrival.

9.3.2 Routing with Current States

In reality, connectors could travel between several cargo supply locations, such as staging base or intermediary port. This requires a process for selecting the loading and unloading nodes to use for the next trip. The selection should be based on the state of the simulation including the amount of cargo at a location and the queues.

The selection algorithm will have to considered each possible combination of load and unload nodes because the cargo demand could be different. For example, the demand for an intermediary port depends on what is needed at the ultimate goal, what is already at intermediate locations and what is in route. For this example, intermediary ports and Sea Base resupply will not be considered for simplicity, resulting in a single demand. The queues will be approximated based on how many connectors are at a loading node compared to the number of load points. The current queue will be approximated as:

$$Queue = \frac{\text{Number of connectors}}{\text{Number of load points}} * \text{Average load time}$$

The loading location and the cargo selected will be determined when departing after unloading based on the travel time, load time, available cargo, and predicted queue.

The amount of cargo selected is tracked so a connector would not be routing to a location which would have no cargo upon arrival. Since each interface at an option would have the same predicted queue, the interface to use is not selected until arrive at the loading node.

9.3.3 Routing with Predicted States

Based on the concept of a threshold, this algorithm predicts the workload on a load interface and routes according to this. Because the routing must occur upon leaving the unloading point, this algorithm must also account for the time spent traveling, so an exact threshold algorithm can not be implemented. It is proposed that improving the routing algorithm will increase the accuracy that it predicts the queues and reduce the overall queue time. By tracking where connectors have been routed, the queues at locations can be predicted into the future to improve the estimations. The detailed process on predicting queues at locations will be discussed in Chapter 10. The basic process is:

1. For each load (pull) and unload (push) location, repeat steps 2 through 8
2. Identify cargo needed at the push location
3. Calculate what can be carried on connector - if no cargo is available or compatible, the push location is removed from possibles
4. Identify possible spots to use based on cargo objects at location - this calculation does not use the current spots, but all spots
5. Identify the fastest predicted wait from the possible spots, add loading time for this connector to the selected spots predicted wait
6. Calculate expected wait time at pull location using predicted waits at possible spots

7. Calculate travel time for current location to pull location then push location
8. Calculate total trip time, travel, waiting, and load/unloading where travel time is subtracted from wait time
9. Identify best pull/push pair which define the next mission
10. Update global predicted wait matrices based on selected push/pull pair

The best pull/push pair is defined as that which maximizes the number of demands serviced per unit time [151]. This will be inverted to a minimization of total trip time to cargo delivered, seen below. The selection of route thus favors shorter trip times and larger cargo load.

$$\frac{\text{total trip time}}{\sum_i P_i x_i}$$

9.3.3.1 Calculation of Predicted Waits

The calculation of expected wait time is based on the equations below. This calculation is based on using the estimated waits for each load and unload point and subtracting off the estimated time to reach that location.

$$\text{time to pull} = \text{pull distance/speed} * 60$$

$$\text{time to pull} += max(pullwaittime - \text{time to pull}, 0)$$

$$\text{time to push} = \text{time to pull} + \text{pull loading time} + \text{push distance/speed} * 60$$

$$\text{time to push} += max(\text{push waittime} - \text{time to push}, 0)$$

$$\text{total time} = \text{time to push} + \text{push loading time}$$

9.3.4 Routing with Predicted States and Retesting

The previous formulation selected the exact interface upon leaving the unloading location, but there may be cases where using a different interface would be an improvement. For example, a loading point could become unavailable or a connector

could not reach the anticipated location, such as in the case of necessary repairs. This is captured by allowing the connector to reconsider its cargo to load and interface selection upon arrival at its loading location. This will not change the routing, but may change the cargo loaded and interface.

9.3.5 Predicted States with Current State Corrections

Incorporating reconsideration causes the vessels to no longer queue for a specific spot, which could impact the predicted queue for the original load interface. To model the impact of this effect on the predictive ability, the previous algorithm was combined with the routing with current states algorithm. The vessels on route have predicted waits for specific load interfaces but the vessels at a location are in a common queue and these two waits are combined to form the predicted waits used for routing. The interface option to use at that location is determined based on the actual queue and cargo conditions at the load point.

9.4 Comparison of Routing Algorithm

The comparisons will be based on a formulation including one type of connector, one type of cargo ship at the Sea Base, one type of beach landing spot, and one stationary cargo supply node. In this case, the number of connectors and number of beach spots will be equal so that queuing at the beach does not impact the results. The supply node and the cargo ships will each have one interface with the stationary node requiring one hour to load and the cargo ships requiring 90 minutes to interface and load. The cargo ships will be located 100 nmi from shore and the cargo supply node 200 nmi. The cargo at each Sea Base ship is 1500 pallets and the cargo supply point has 3000 with the total beach demand of 3000 pallets.

To compare the waiting time for each algorithm, one cargo ship, one supply point, and two connectors are considered. The basic formulation is shown in Figure 18 with one connector starting at the supply base and one at the cargo ship. A third

connector is added to see how queuing impacts the routing, starting at the cargo ship. The dashed section of the figure is the addition to test the robustness to disturbance. A second cargo ship is added that is available for the first 12.5 hours of operations and then unavailable for 12.5 hours. Three connector will be used to maintain the number of load interface equal to the number of connectors with the possibility of queuing. For each comparison, the algorithms will be run for 10 simulation days.

The forth and fifth comparison will be to see the impact of the ratio of travel to load time. First, the distance to the Sea Base is increased to 250 nmi and the supply point to 1000 nmi. The loading time is 2.5 hours for the supply point and 5 hours for the Sea Base cargo ship. Two cargo ships are included and three connectors, to make reconsideration of interface a possibility. The Sea Base ships are then moved to 50 nmi taking 10 hours to load and the supply point set to 200 nmi, taking five hours to load.

9.4.1 Basis for Comparison

The routing algorithms will be compared based on comparing total time to deliver as well as ability to predict trip times. This will include comparing time histories of the cargo delivered and the total shortfall. The concept of shortfall was introduced in section 5.5 and for this example will be calculated for every hour, in essence representing the integral between the cargo delivered and the cargo demanded. A lower value is desirable. The final plot will present the time history of the cumulative difference between the predicted trip time and actual trip time.

9.4.2 Comparison 1

This scenario is the most basic formulation with two connector, one cargo ship at the Sea Base and one supply point. This scenario was run for the five routing algorithms. The time to complete is given in Table 9 with the No Routing algorithm failing to complete within 10 days. Figure 19 shows the amount of cargo, in the percentage

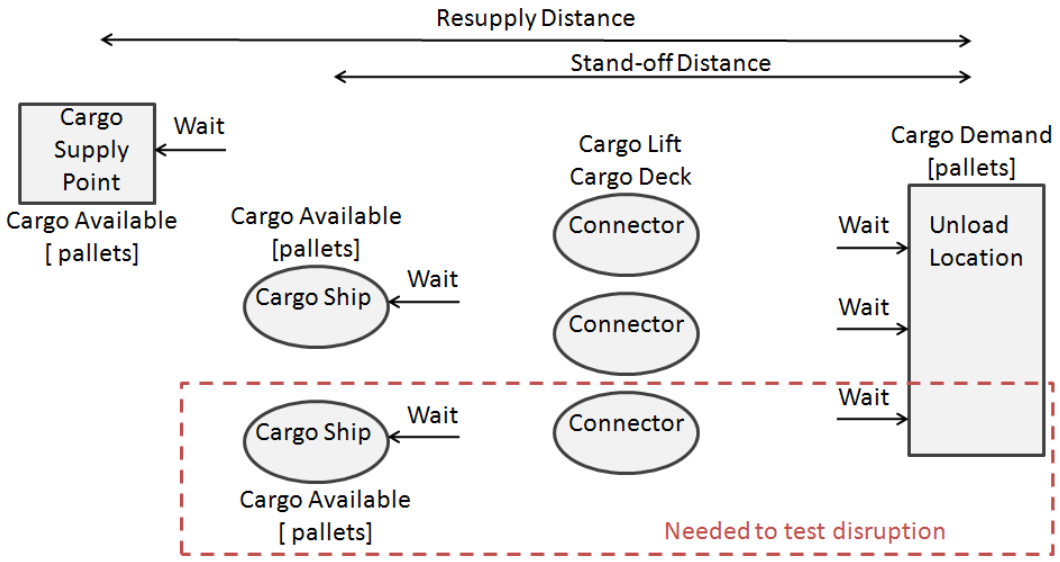


Figure 18: Example Problem Formulation

Table 9: Summary Comparison Results - Time to Complete

	Time To Complete (days)				
Algorithm	Comp 1	Comp 2	Comp 3	Comp 4	Comp 5
No routing	9.63	9.63	3.67	9.71	5.92
Basic Routing	7.25	5.00	3.29	8.46	6.63
Predictive Routing	7.25	5.00	3.42	8.46	5.92
Pred with Reconsideration	7.25	5.00	3.29	8.46	5.92
Corrected Pred with Reconsideration	8.04	5.46	4.29	-	6.21

of pallets delivered. This gives more detailed information that the time to complete as it shows the basic routing, predictive routing and predicted with reconsideration all yield the same time history. This overlay of results is also seen in Figure 20. Figure 21 plots the cumulative difference between the actual trip time and predicted trip time. A positive value represents higher actual trip times than predicted. The no routing algorithm predicts the trip time perfectly, which is understandable since with each connector traveling between a load point and unload point, no queuing would occur and thus no additional waiting. The predicted and predicted with reconsideration perform the same in this example as there is only one load point at each location. These algorithms are very close in estimation, although the estimate is slightly low near the beginning of the simulation. The basic routing algorithm greatly overestimated the total trip time, although not as poorly as the corrected predictive algorithm. This over prediction is due to the algorithm not considering vessels being loaded during the time the vessel is traveling to the destination. The predictive with correction further overestimates because it adds the vessels in transit to the wait time, increasing the time estimate.

9.4.3 Comparison 2

Adding an additional connector improves the performance except for the no routing case as seen in Figure 22. The no routing algorithm is not improved because it can not take advantage of the additional connector since it is set to travel between the Sea Base and shore, and its prediction is off by the queue time of one ship, after which the two connectors at the Sea Base are separated by enough time and distance that they do not queue again. One connector continues to travel to the supply point even after cargo runs out at the Sea Base. This impact is also seen in Figure 23, with the no routing shortfall greatly exceeding the other algorithms. Once again, the other algorithms perform generally the same, with the corrected predicted algorithm

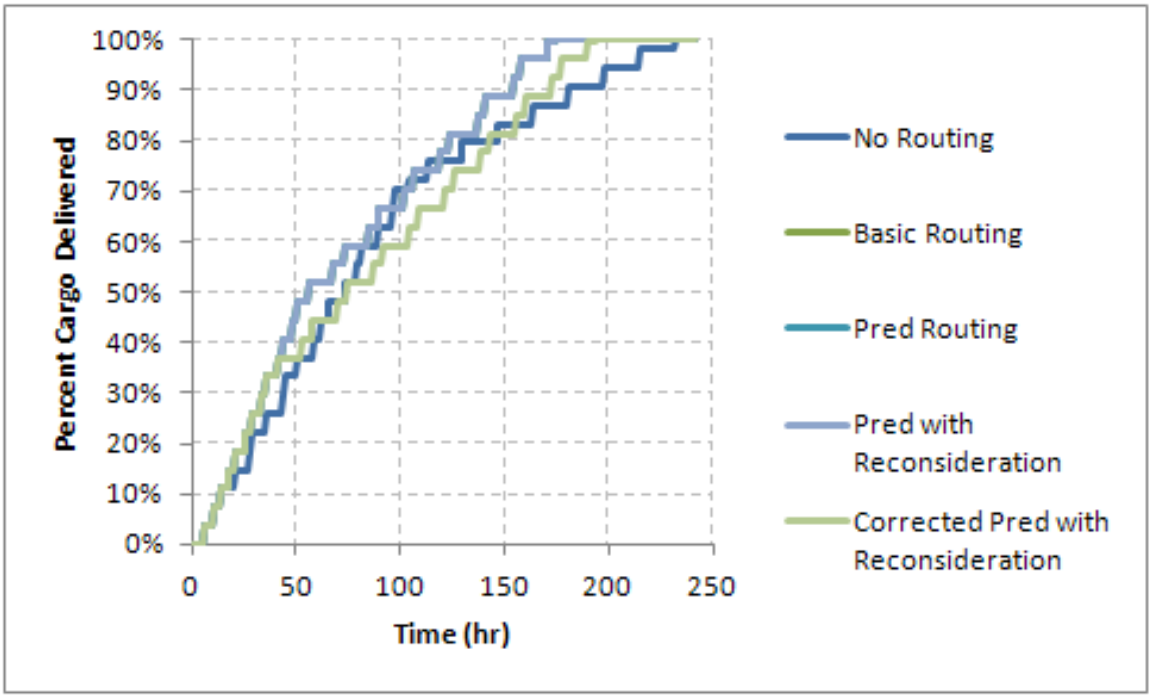


Figure 19: Comparison 1: Time History

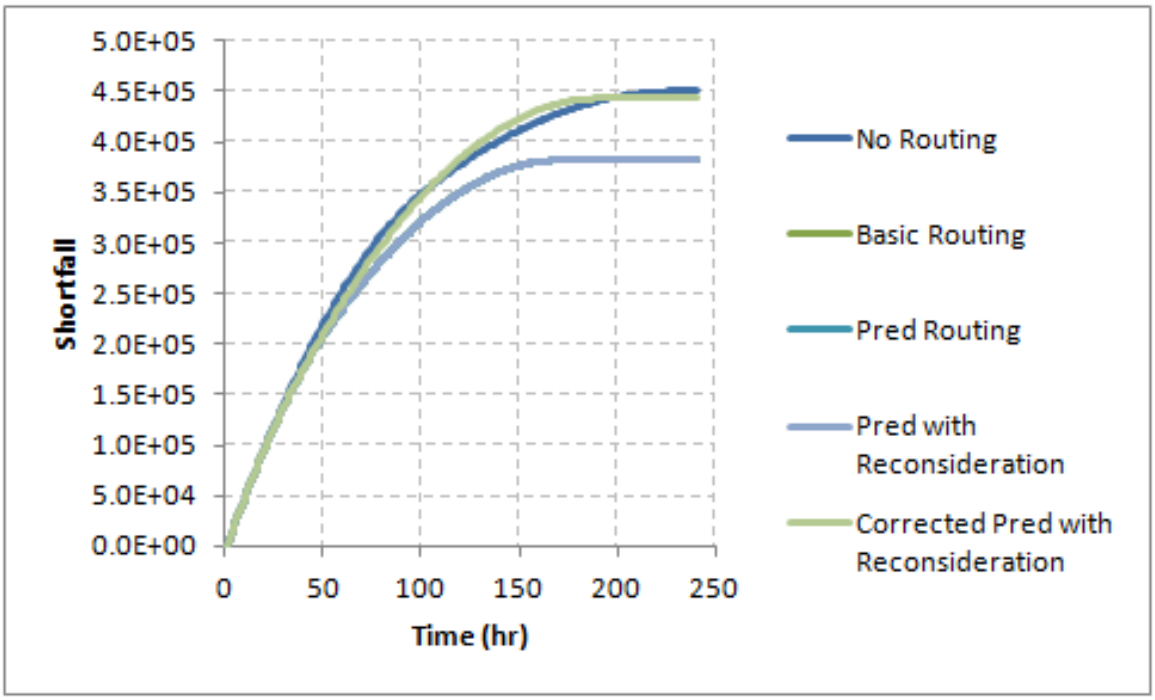


Figure 20: Comparison 1: Shortfall

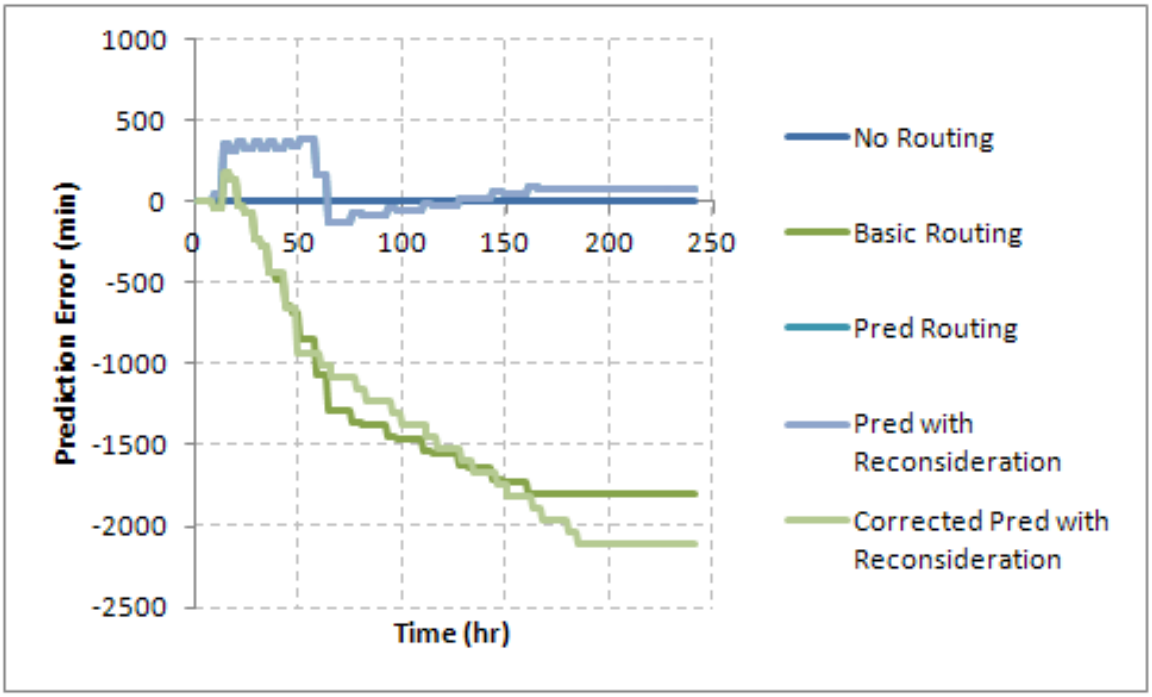


Figure 21: Comparison 1: Difference between Predicted and Actual Trip Times

lagging slightly. In Figure 24, the no routing is again seen to predict best since it does not have any queuing in its formulation. The remaining algorithms over predict the total trip time with the predicted routing and predicted with reconsideration yielding the best estimation.

9.4.4 Comparison 3

This comparison adds a disruption in the loading at the Sea Base with the unavailability of an additional cargo ship. This ship is only unavailable for a short amount of time and doubles the amount of cargo available at the Sea Base so the time to complete is faster in all cases as seen in Figure 25. The weakness of the basic routing and corrected predicted routing is seen in the prediction of the trip time in Figure 27. No routing still predicts well as queuing only occurs when one ship is unavailable and this queuing spaces out the two connectors so it does not occur again as the travel and unload time are greater than the time to load. Predicting with reconsideration

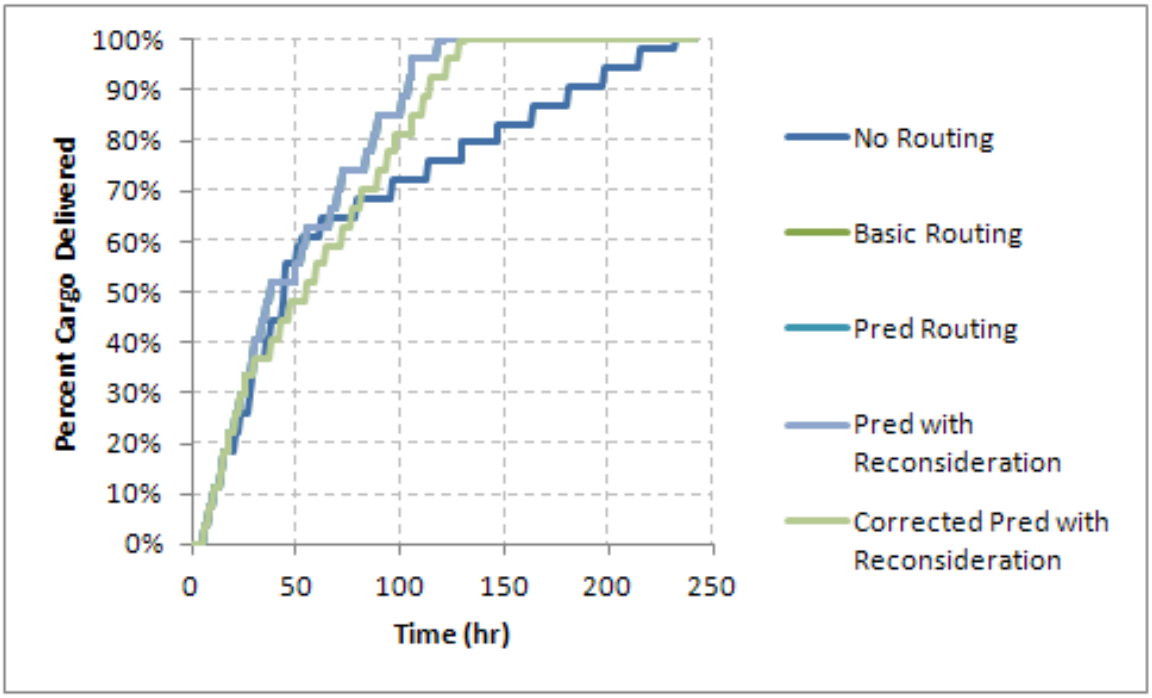


Figure 22: Comparison 2: Time History

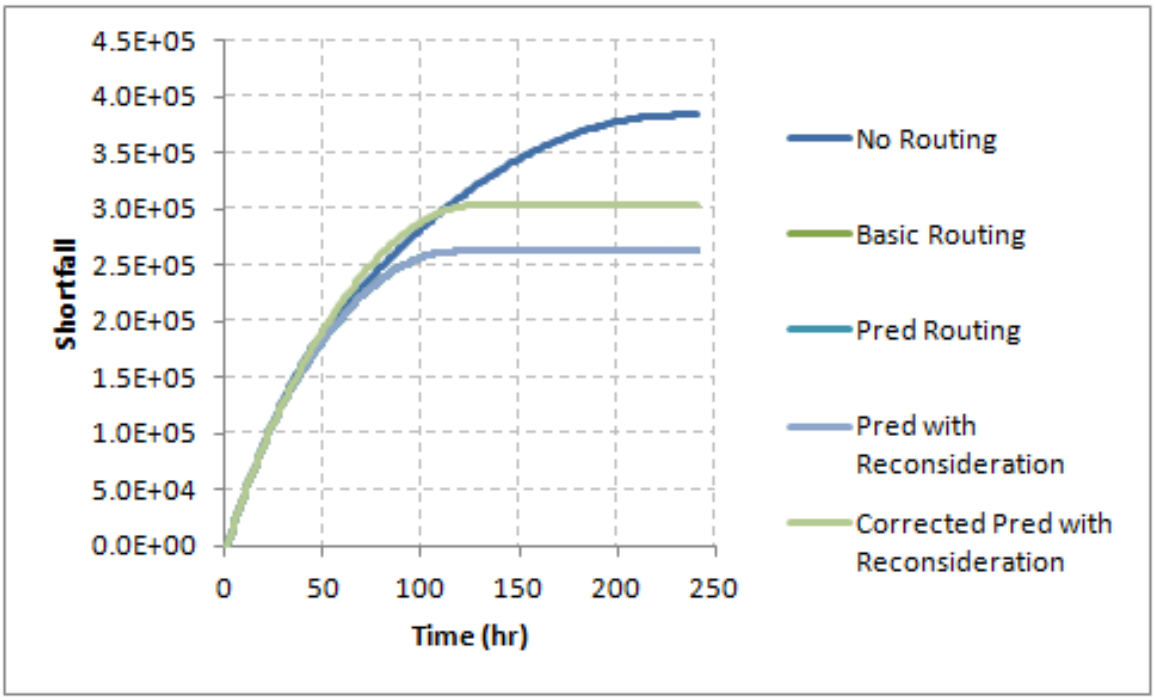


Figure 23: Comparison 2: Shortfall

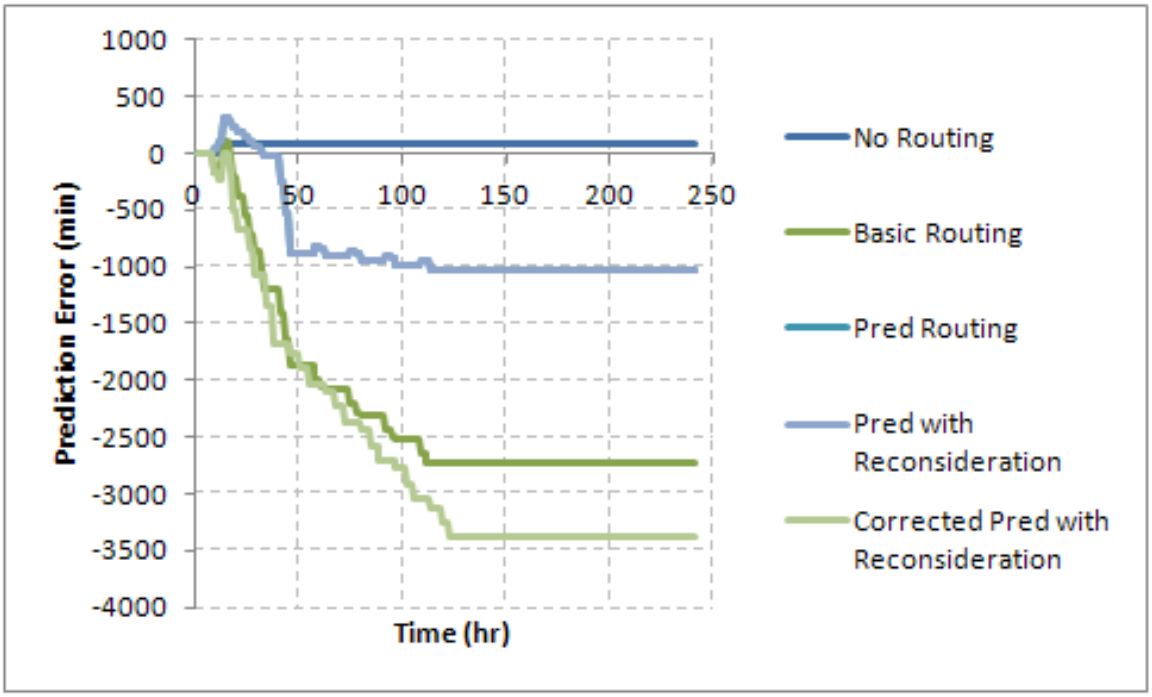


Figure 24: Comparison 2: Difference between Predicted and Actual Trip Times

does slightly better than only predictive because the predictive algorithm could not adjust to the missing connector as quickly since it could not change the load point to use once the ship left the beach, seen in the gap in the shortfall level in Figure 26. If a connector departed the beach while the ship was present but arrived during its unavailability, this connector would be unable to be loaded at the other ship even if it was available, creating additional queuing and driving up the error in prediction of total trip time. The basic queuing also included reconsideration, in that the actual connection point is not selected until arrival at the load point, thus it is not surprising this algorithm performs well with a disturbance.

9.4.5 Comparison 4

The fourth comparison moves the ships and supply point further from the delivery point to test the algorithms performance when travel time is much greater than loading and unloading time. This slows down the delivery as seen in Figure 28 and

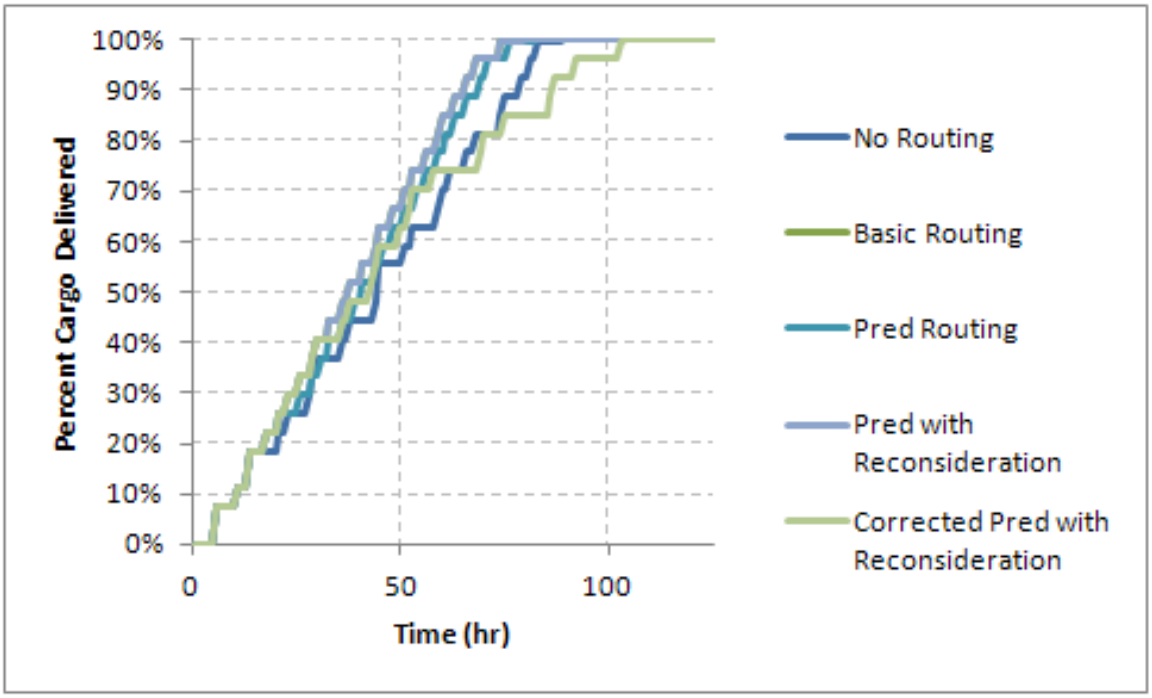


Figure 25: Comparison 3: Time History

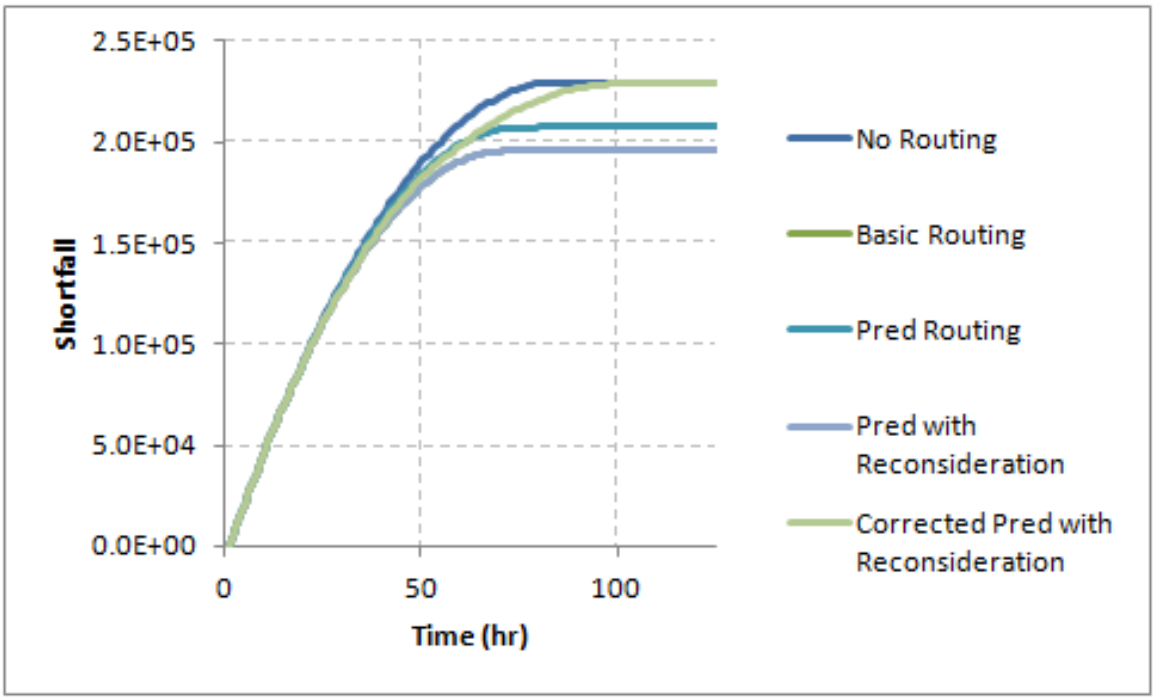


Figure 26: Comparison 3: Shortfall

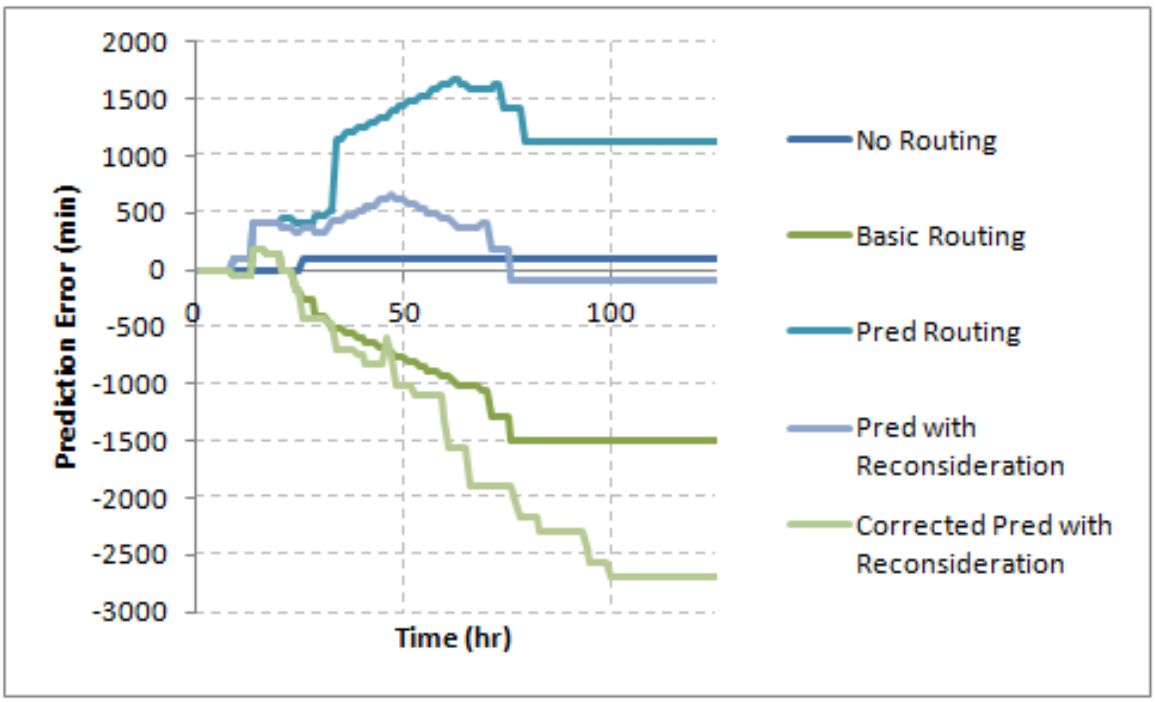


Figure 27: Comparison 3: Difference between Predicted and Actual Trip Times

gives further insight into the algorithms weaknesses. The corrected prediction with reconsideration seems to make a poor selection, separating itself from the basic routing by sending all the connectors to the supply point. This is most likely due to its poor prediction of trip time as seen in Figure 30. This algorithm does not complete delivery during the simulation time and the shortfall would continue to increase, exceeding the shortfall of the no routing algorithm, as seen in Figure 29.

9.4.6 Comparison 5

Decreasing the distance to the Sea Base and load point makes the total travel time less than the loading time at the Sea Base. Figure 31 illustrates the predictive routing algorithms and no routing perform the best and the basic routing algorithm has the greatest time to complete delivery. This performance is mirrored in the shortfall, Figure 32. Figure 33 shows the basic routing underestimates the total time because it does no account for the vessels in travel and corrected prediction continues to

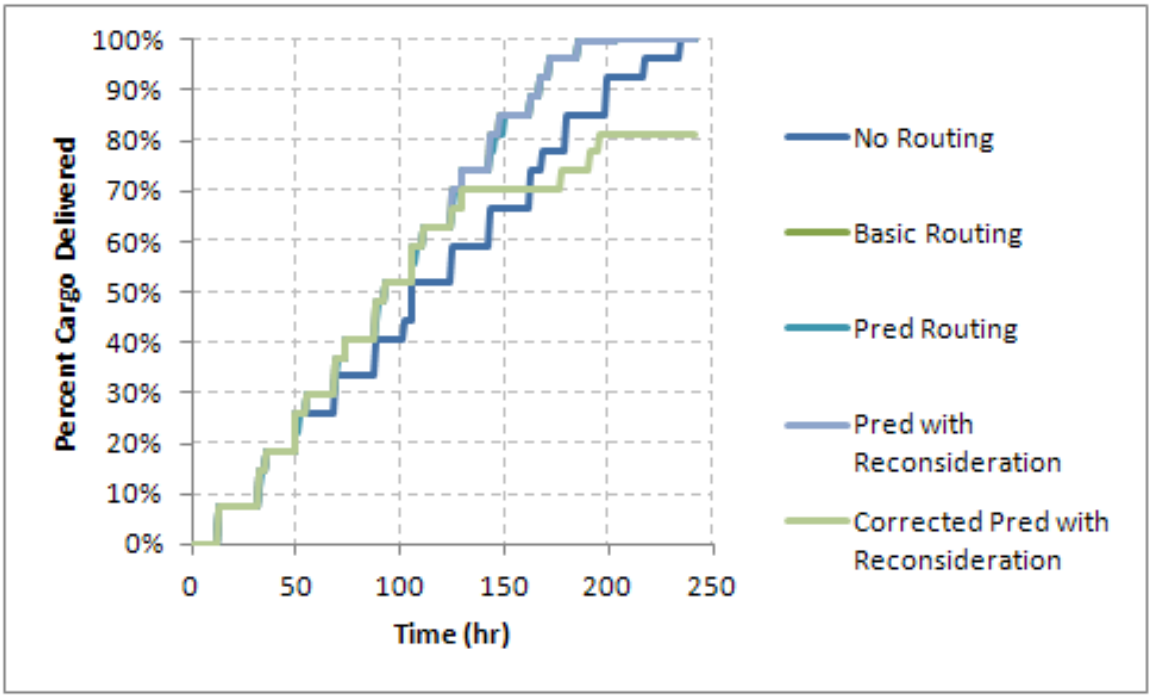


Figure 28: Comparison 4: Time History

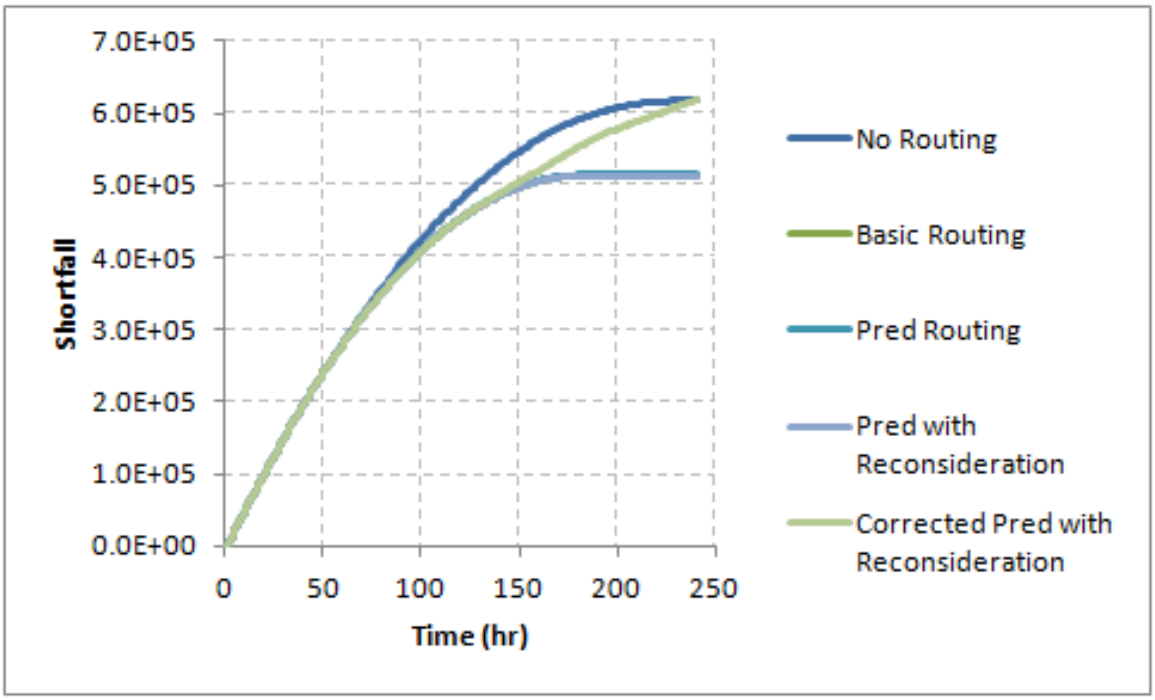


Figure 29: Comparison 4: Shortfall

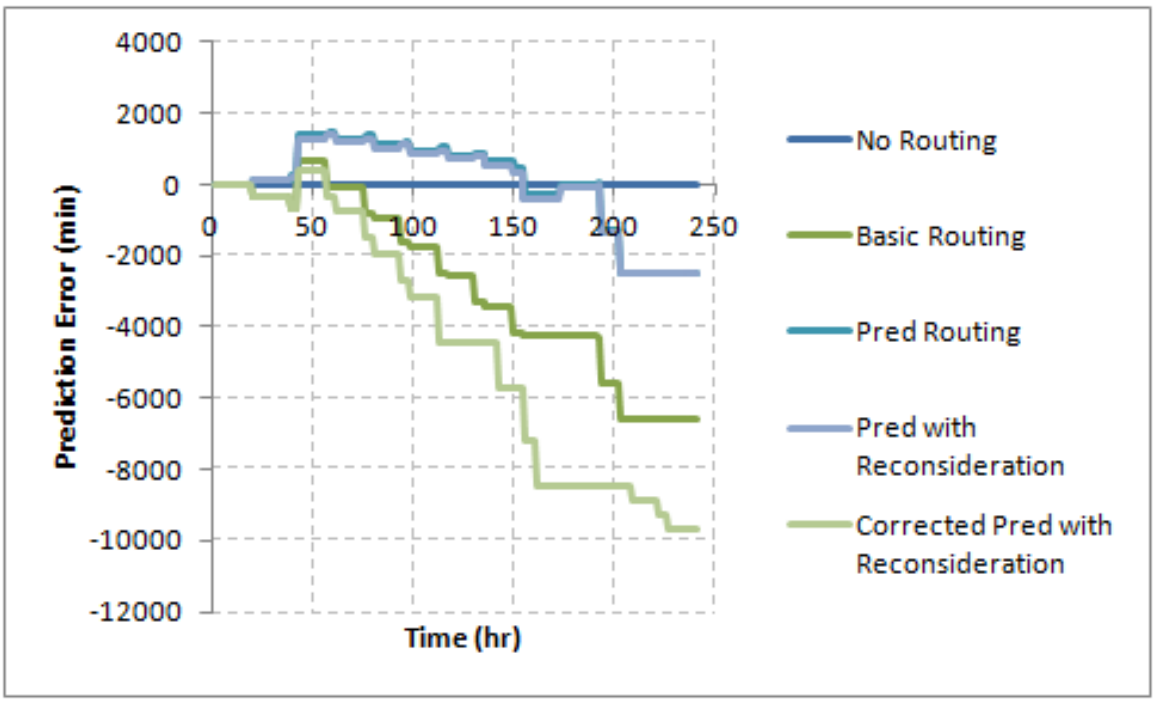


Figure 30: Comparison 4: Difference between Predicted and Actual Trip Times

overestimate the total trip time.

9.4.7 Stochastic Tests

Thus far, all of the comparisons have been for deterministic, but in reality, the load times would not be a single number. Although not widely considered in this thesis, the algorithms will be compared using the formulation for Comparison 4 and 5, but with a distribution on the wait time. The average wait time will be used for the routing algorithm, but the actual wait time will be a triangular distribution varying 20 percent in both directions.

9.4.7.1 Comparison 4

Running 250 repetitions of the Comparison 4 scenario, the distributions for the time to deliver and total error in prediction of trip time are plotted in Figure 34, Figure 35 and 36; showing the stability and quality of the algorithms. Note the corrected predicted routing does not complete the delivery for any case so it is not included in

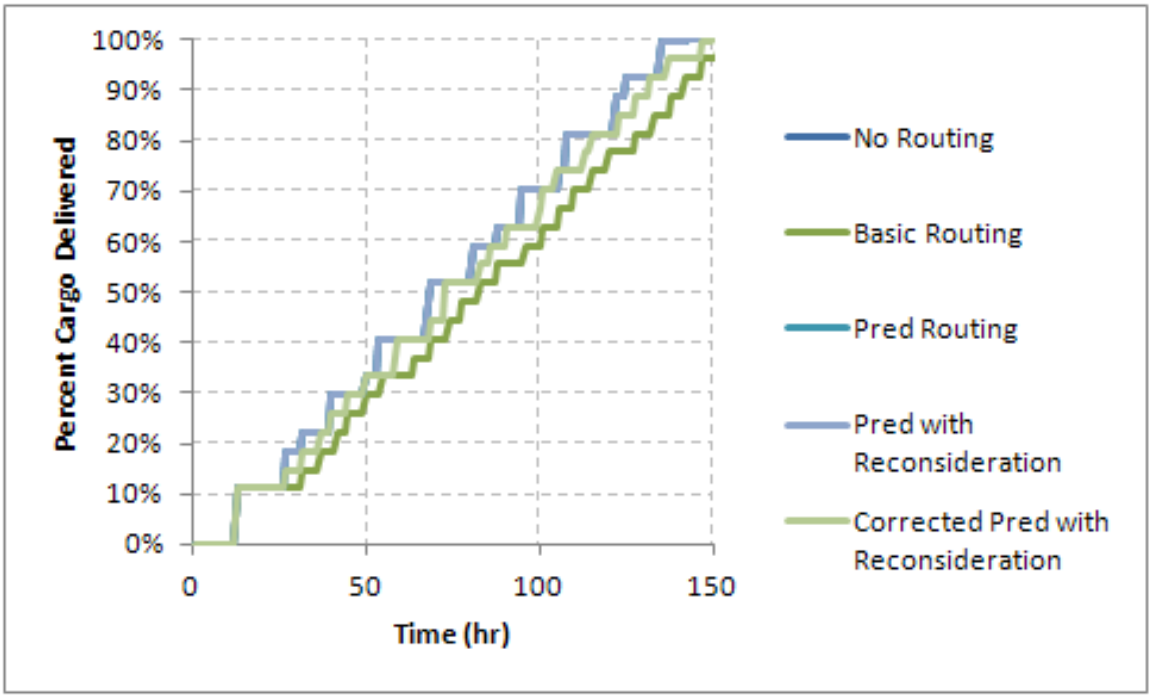


Figure 31: Comparison 5: Time History

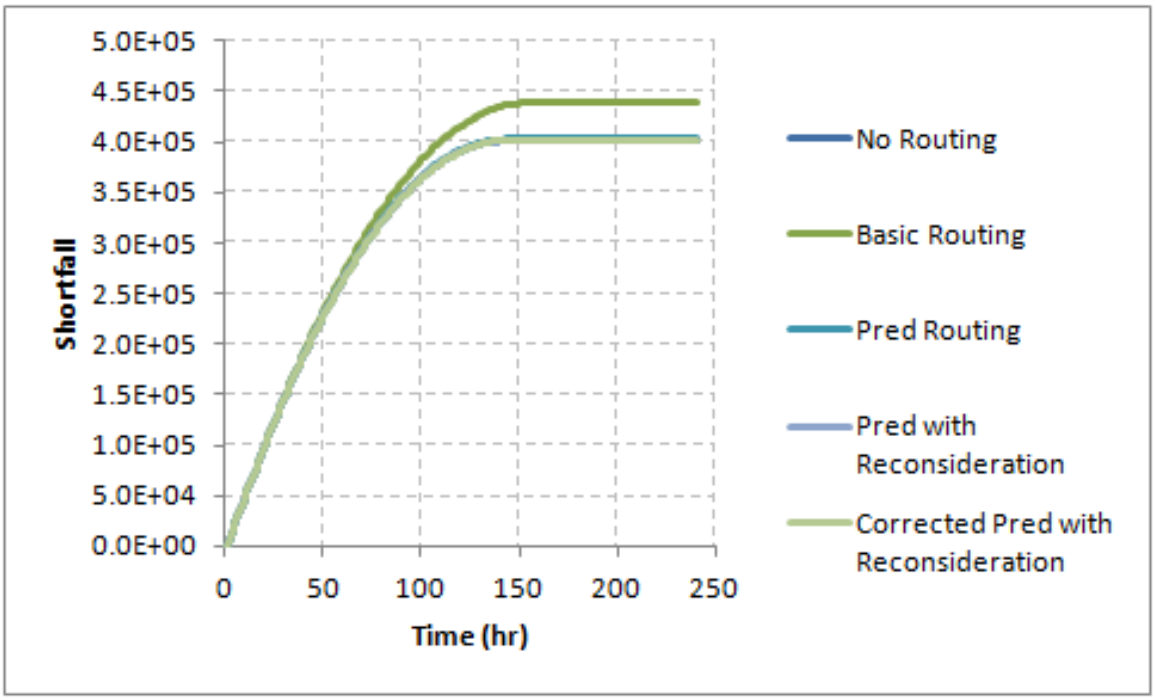


Figure 32: Comparison 5: Shortfall

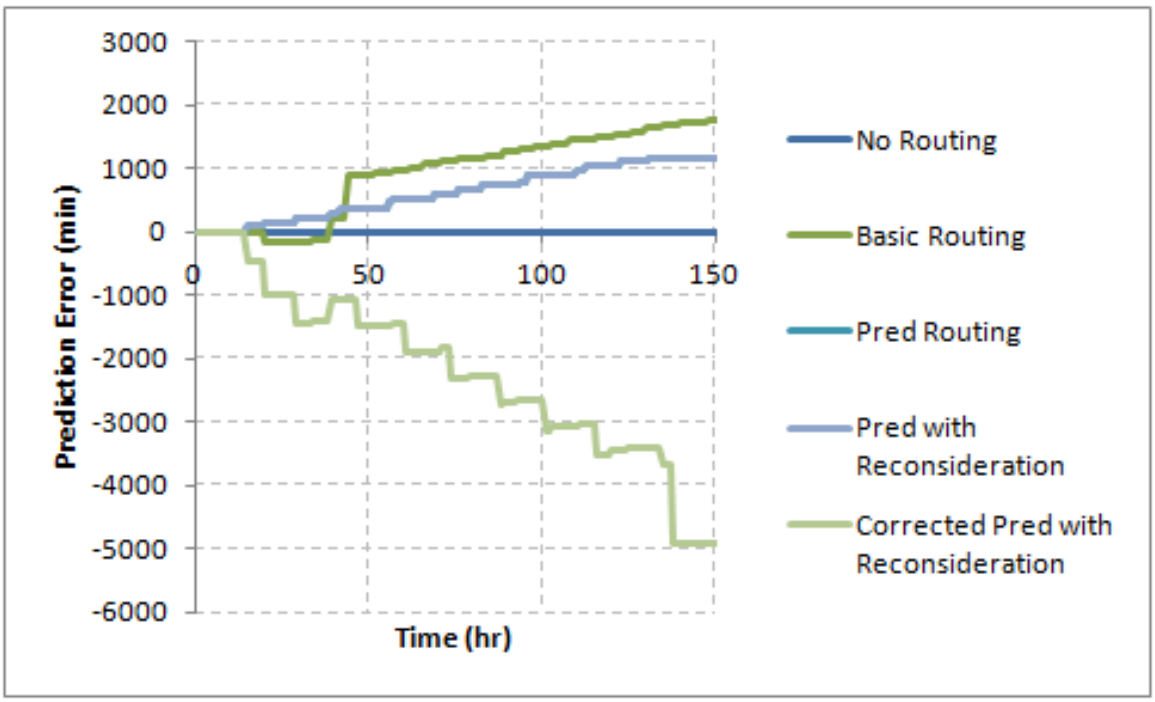


Figure 33: Comparison 5: Difference between Predicted and Actual Trip Times

Figure 34. The shortfall for this algorithm would continue to increase if the simulation time was expanded, while the other algorithms have reached the demanded cargo level so no additional shortfall would accumulate with additional simulation time. No routing incurs the least total error but takes the longest to complete for all cases. The distribution for basic routing and predictive routing with and without reconsideration perform with the same general range. Without reconsideration is shifted slightly to the right with a slightly longer time to complete. All three of these algorithms have a small range, so the algorithms are stable.

9.4.7.2 Comparison 5

Two hundred and fifty repetitions were run for the scenario in Comparison 5 and the distributions are shown in Figure 37, Figure 38 and Figure 39. The predicted routing with and without reconsideration are the most consistent in their prediction time error and the time to complete. With reconsideration preforms slightly better than

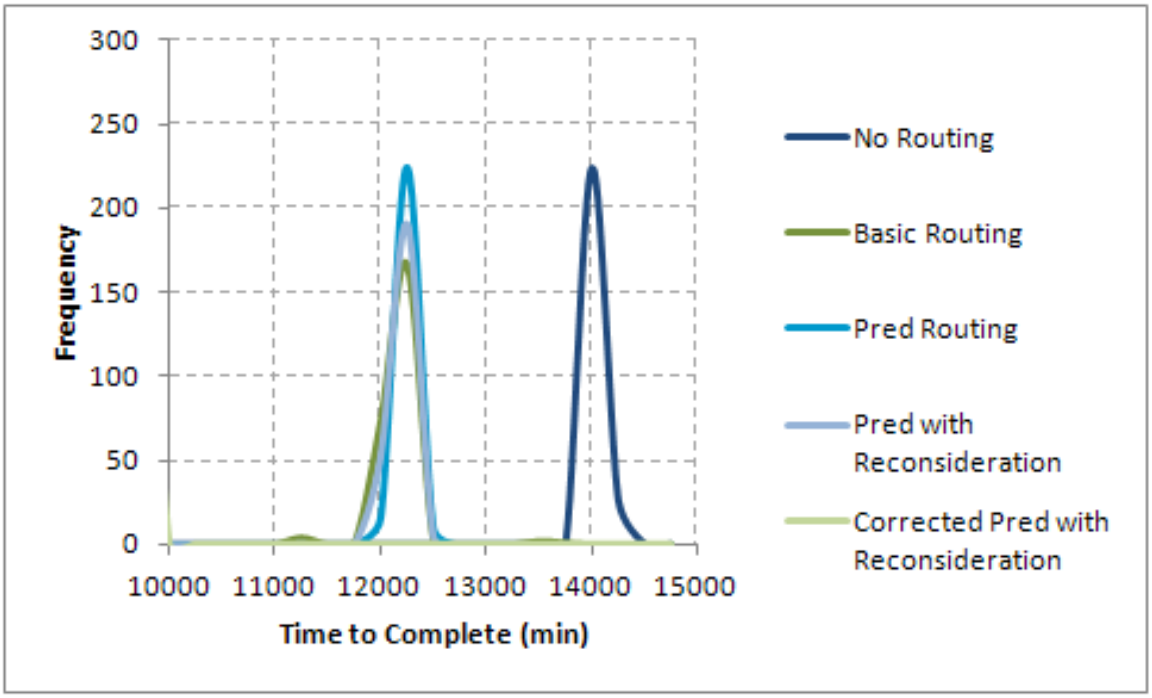


Figure 34: Distribution of Time to Complete

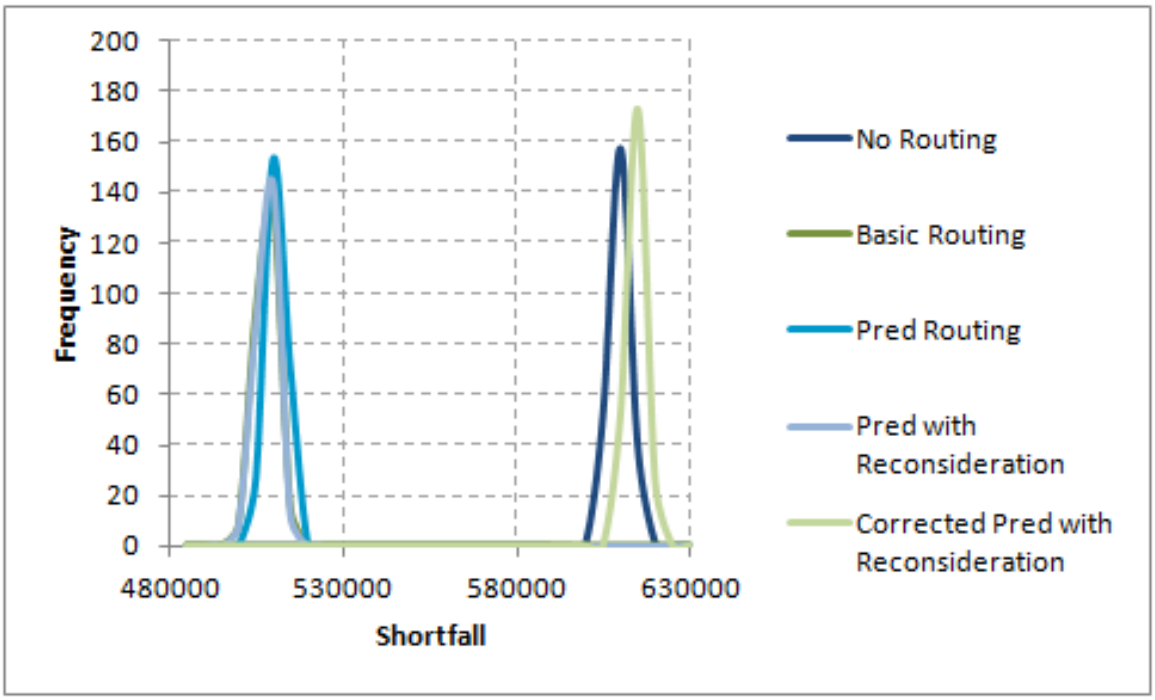


Figure 35: Distribution of Shortfall

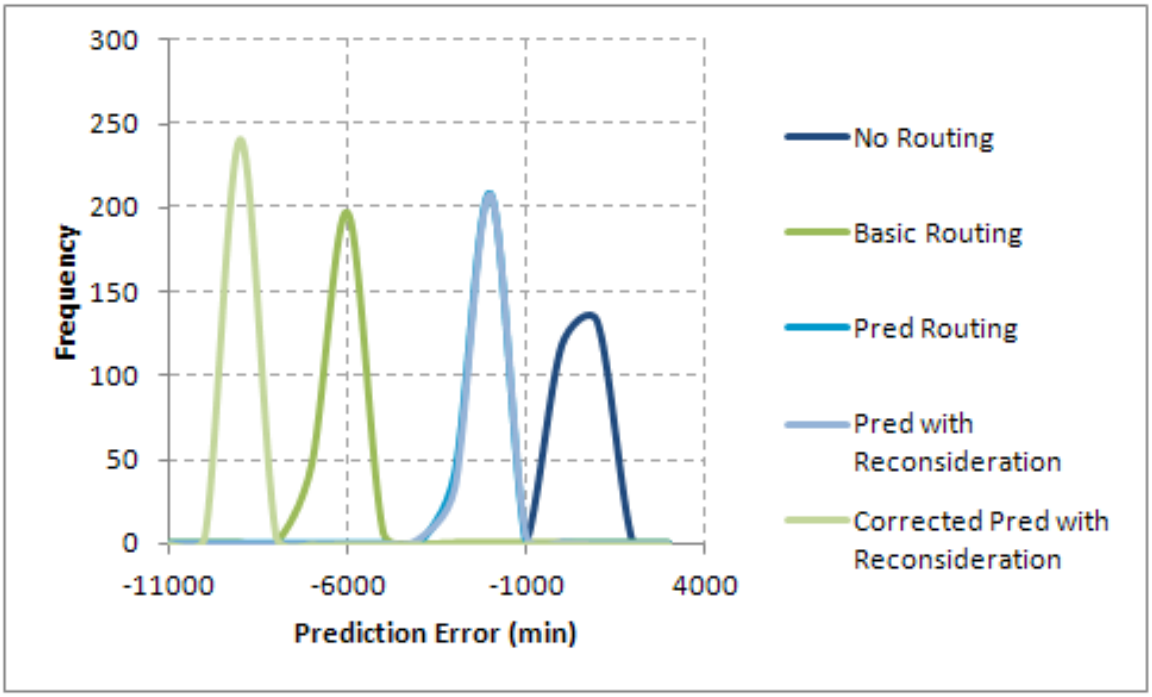


Figure 36: Distribution of Difference between Predicted and Actual Trip Times

without, but both are better than the basic routing and corrected prediction, which incorporates the basic routing calculations. No routing performs the best because the case selected was simple to manually route the vessels. In a larger case, it would be impossible to select the routes that gave the best results, but in this case it provides a good comparison to examine the variability introduced with a dynamic algorithm, as seen in the increase in spread of the time to complete.

9.5 Scalability

To demonstrate their routing capabilities, each algorithm was modeled for a simple case, but it is important that the algorithm is scalable to a large scale model. The challenge here is the increase in nodes and connectors. As was discussed in the motivation, no routing is not a viable option as it is undesirable to have to determine in advance the connection to make. Specifically, this algorithm would perform very poorly if a limited amount of cargo was available at one node.

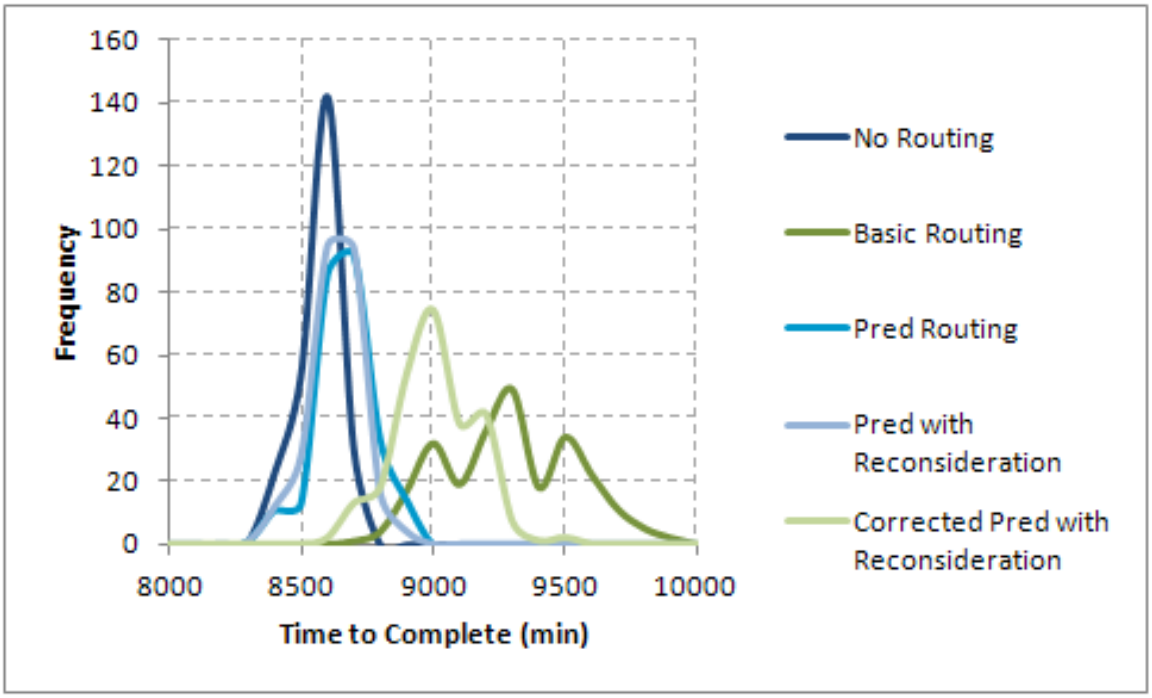


Figure 37: Distribution of Time to Complete

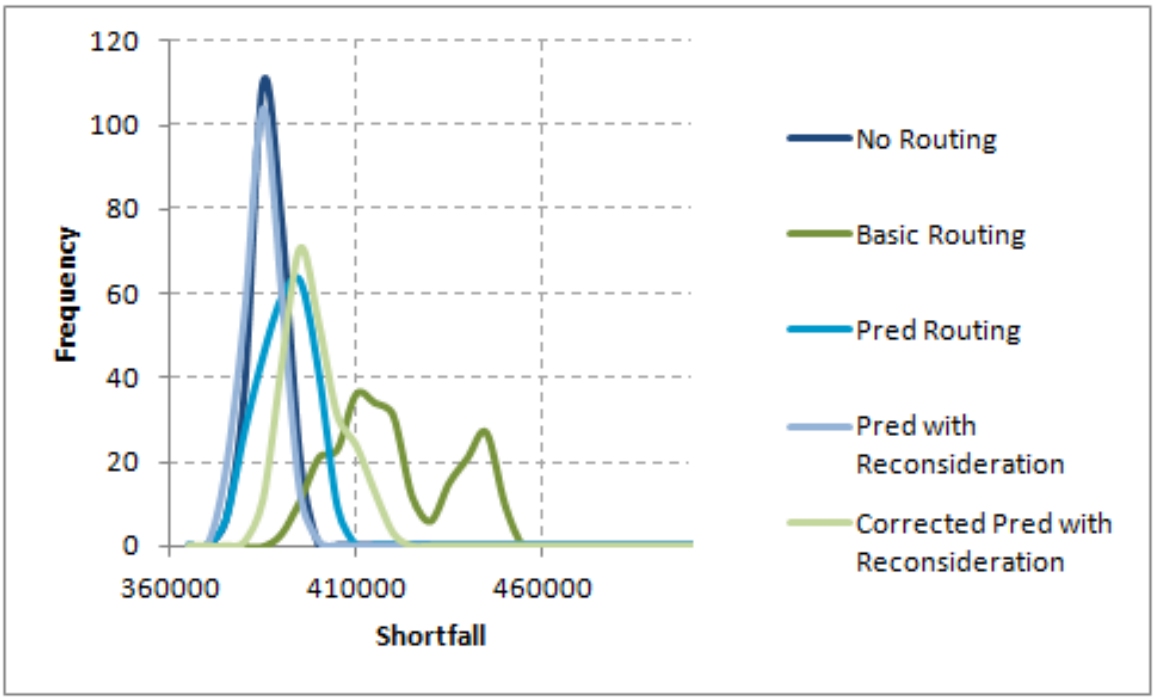


Figure 38: Distribution of Shortfall

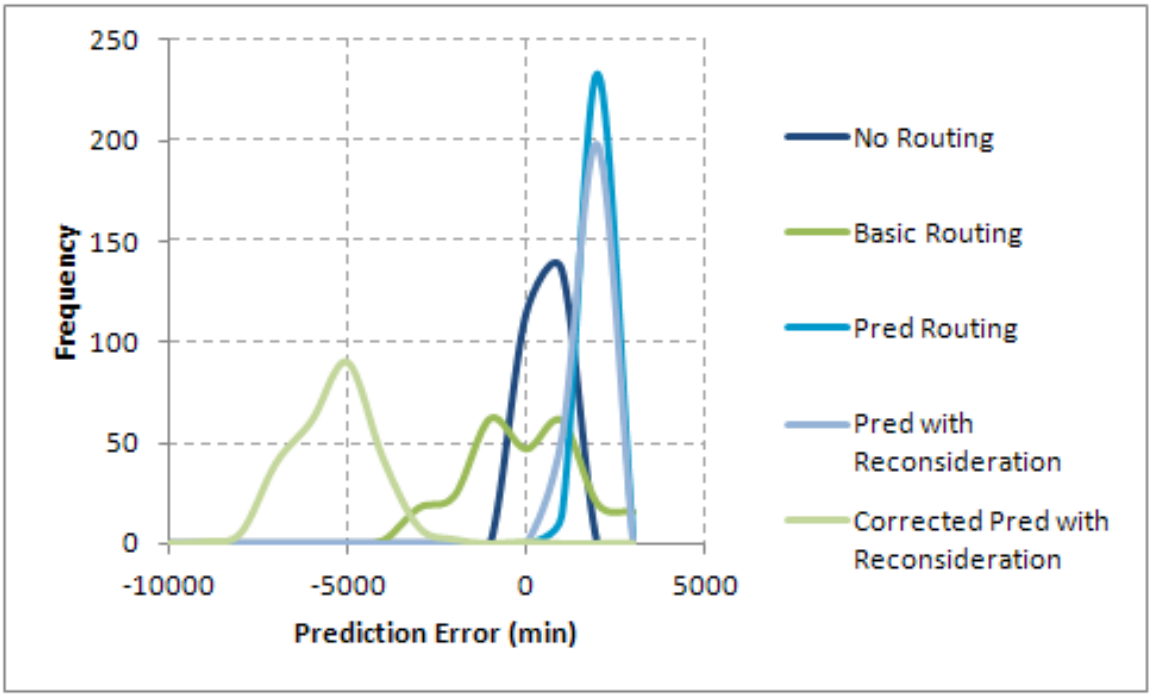


Figure 39: Distribution of Difference between Predicted and Actual Trip Times

The use of the current state of the model is not a scalable option. As different types of interfaces are included, the question would arise as to how to calculate the current queue. Should the interface times for each type of interface be averaged? This is further complicated when it is considered that some interfaces on a single cargo object could be incompatible and this not usable at the same time. While the estimation of this average queue length is still possible, it becomes computationally intensive as the number of types of vessels and cargo locations is increased. Each time a route is selected, it is necessary to calculate the number of connectors and cargo objects at each geographic location. This would need to be repeated for every type of connector as the wait times for different types could vary. This further complicates the calculation of the estimated queue as different types of connectors could use the same interface and the algorithm would have to be expanded to incorporate each type of vessel and their relative wait time. The calculation for this algorithm would grow proportional to the number of vessels multiplied by the number of geographical

locations.

The use of predicted states does not have the algorithm development issues seen with current state algorithms. Each interface is handled separately, so the ability to use different interfaces and the use of an interface by different connectors does not require additional computations. This method does require the maintenance of a global matrix of size n x m , where n is the total number of connectors and cargo nodes and m is the number of interface options in the model. The matrix does not grow with the number of geographical locations, as the location of each object is an existing array in the model.

Overall, the use of a predicted state algorithm is more scalable as the calculation method does not have to be modified with the modification of the vessels included in the model. The predicted state algorithm more fully meets the initial goals of a dynamic routing algorithm in order to have the routing be able to incorporate changes in the architectural structure of the model without modification.

9.6 Selection of Routing Algorithm

It is desired to select a routing algorithm that does not introduce additional wait time into the performance of the simulation. A series of simulations were performed to compare five possible algorithms by tracking the overall performance and compare the estimated trip time compared to the actual trip time. While the baseline case of preset routings could accurately predict the trip times, it introduced lag into the overall performance and does not create a flexible algorithm that can be expanded to a real size problem. The expandability and trip prediction capability are lacking with the use of current states as the algorithm or to correct the predictions. Taking into consideration all of the comparisons, the predictive algorithms performed the most consistently and the predictive algorithm with reconsideration is most able to handle and estimate in the case of disruptions. The inclusion of a distribution on the

wait times identified the strength including reconsideration with taking advantage of changes that can speed the loading process. Additional probabilistics, such as repairs, would further improve the performance of the predictive with reconsideration algorithm. This algorithm is scalable to a heterogeneous mix of vessels of any size and number of supply nodes.

9.7 Special Case - Multiple Unload Points

As the size of the logistics craft increases, a single load of cargo may be more than a single beach requires. In this case, hopping between beaches may be advantageous, fulfilling the cargo requirements at two or more beaches in a single trip. Selecting the hops is difficult because the trip will take longer but would have a fuller vessel and the number of hops to make depends on the cargo demand. To select the route between shore nodes, a traveling salesman problem (TSP) will be combined with the existing dynamic loading algorithm.

The traveling salesman problem is a classical formulation for routing by selecting nodes to visit. One TSP is prize collecting, where only profitable nodes are visiting with an edge cost incurred for traveling between nodes [123]. The goal is to leave a home location, visit a sequence of locations and return to the home location while incurring the lowest cost [46]. It is not necessary to visit all of the available locations, so this is a traveling salesman subtour problem, which is still shown to be NP-hard [197]. To capture this problem, n(n-1) variables, where n is the number of locations to visit, are needed to represent the connections between the locations in addition to the n variables for the locations to visit [142]. This problem has been shown to be solvable using branch and bound for up to 300 verticies [84] so the complexity will not be an issue for scenarios of realistic size for this problem.

The general requirements for setting up a knapsack constrained TSP was described by Tang and Wang [177] and has been modified from a penalty paid for unvisited

customers formulation. The objective is to minimize the travel costs while maximizing the profit. This is subject to the assignment problem of locations to visit and a constraint that each location can be visited at most, once. Additional constraints require the depot to be included in the route and eliminates subtours. Knapsack-like constraints enforce a minimal profit and the maximum capacity of the salesman. This formulation leads to the conclusion that the TSP formulation can be added to the knapsack and assignments problem used in the dynamic loading algorithm.

9.7.1 Objective Function

The objective function must balance the profit of the cargo delivered with the travel cost of the trip. To use a branch and bound formulation, the objective function is a linear combination of the variables [84] or ones of the objectives can be constrained with a specific bound [74]. Both of these works suggests a weighted linear combination of the form below where M is sufficiently large, $p_k x_k$ is the profit of the verticies visited and c_{ij} is the cost of the traveled leg x_{ij}.

Maximize

$$M\sum\nolimits_k p_k x_k - \sum\nolimits_{i,j} c_{ij} x_{ij}$$

For this work, the priority, p_k, it already normalized and c_{ij} is in minutes. To make these two values compatible in one equation, the total time cost will be normalized for the shortest possible delivery trip. This compares the relative priority to the relative trip time:

Maximize

$$\sum\nolimits_k p_k x_k - \sum\nolimits_{i,j} \frac{c_{ij} x_{ij}}{min(c_{ij})}$$

9.7.2 Formulation of Linear Program

This formulation combines the dynamic loading algorithm with the traveling salesman constraints. The demand constraint used in the loading will be modified for the total

demand to a demand constraint based on the beach unload locations to visit. The decision variables are:

x_i amount of cargo type i loaded

$$y_j = \begin{cases} 0 & \text{if not using load point j} \\ 1 & \text{if using load point j} \end{cases}$$

$$z_k = \begin{cases} 0 & \text{if not using beach k} \\ 1 & \text{if using beach k} \end{cases}$$

$$n_{m,p} = \begin{cases} 0 & \text{if not traveling from m to p} \\ 1 & \text{if traveling from m to p} \end{cases}$$

$c_{m,p}$ cost of traveling from m to p

To form the objective function, the priority of the cargo is weighted against the time to complete the trip. The cost of travel includes an estimated wait time for each beach location added to the time to unload at that location. The complete LP formulation becomes:

Maximize

$$M\sum_i P_i x_i - \frac{\sum_m \sum_p c_{m,p} n_{m,p}}{minimum(c_{m,p})}$$

subject to:

$$\sum_i weight_i x_i \leq maxlift$$

$$\sum_i area_i x_i \leq maxarea$$

$x_i \geq 0$ for all i

$x_i - \sum_j cargo_{j,i} y_j \leq 0$ for all i

$$\sum_j y_j = 1$$

$x_i - \sum_k demand_{i,k} z_k \leq 0$ for all i

$\sum_m n_{m,p} \leq 1$ for all p

$\sum_p n_{m,p} \leq 1$ for all m

$$\sum_m n_{m,0} = 1$$

$$\sum_p n_{0,p} = 1$$

$\sum_p n_{m,p} - \sum_m n_{m,p} = 0$ for all m,p

$n_{m=p} = 0$ for all m,p

$n_{m,p} + n_{p,m} \leq 1$ for m,p ≥ 1

$z_k - \sum_m n_{m,p} = 0$ for all p

Where the first three equations are the knapsack problem. The next two are the load assignment problem. The next equation is the demand based on the beaches selected. The last eight constraints are the traveling salesman, where not all the beaches must be visited, but the trip must start and end at beach 0, which is a dummy depot. The connector can not return to the same beach and may not have subtours. The last equation resolves the trips selected to beaches visited.

9.7.3 Small Example

To demonstrate the execution of this LP, a small example that is solvable by hand is executed. A single type of cargo must be delivered to three beaches. The demand for beach 1 is 1, beach 2 is 4 and beach 3 is 3. It is desired to solve which of these beaches should be visited and how much cargo is loaded from one of the two available supply ships. The cargo ships each have 5 units of cargo available. The cargo weights 10 LT and requires 5 sqft to transport with the connector having a capacity of 100 LT and 100 sqft.

$c_{m,p}$ combines the time to travel between the locations, estimated queues, and the time to unload at that beach.

$$c_{m,p} = \text{ max(travel time, predicted queue)} + \text{ time to unload}$$

For this example, no additional queue time is added

$$c_{m,p} = \underset{\text{Travel Time}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 20 \\ 0 & 10 & 0 & 15 \\ 0 & 20 & 15 & 0 \end{pmatrix}} + \underset{\text{Unload Time}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 \\ 0 & 15 & 15 & 15 \\ 0 & 18 & 18 & 18 \end{pmatrix}} = \underset{\text{Total Time}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 10 & 20 & 30 \\ 0 & 25 & 15 & 30 \\ 0 & 38 & 33 & 18 \end{pmatrix}}$$

This results in transporting 5 cargo items and visiting beach 1 to deliver 1 and beach 2 to deliver 4. The order is inferred from the traveling to row to column. The dummy beach 0 requires the journey to start at stop at that location, so $n_{m,p}$ is read starting in the first row:

$$c_{m,p} = \begin{pmatrix} & Beach0 & Beach1 & Beach2 & Beach3 \\ Beach0 & 0 & 1 & 0 & 0 \\ Beach1 & 0 & 0 & 1 & 0 \\ Beach2 & 1 & 0 & 0 & 0 \\ Beach3 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The demand for beach 1 was increased to beyond the maximum capacity of the connector and the supply at each ship increased to 15 units, the algorithm selects to only visit beach 1, delivering a full load. Returning to the original set-up, the algorithm was repeated for larger time penalties, having the beaches require 100 min to travel between, but keeping the same loading times. This resulted in only beach 2 receiving a delivery of 4 cargo items, due to the large travel times outweighing the cargo that would have been delivered to more beaches. These simple tests have shown the hopping algorithm makes logical decision on the selection of cargo to load based on the relative trip lengths and the amount of cargo that can be carried.

9.7.4 Modification to Overall Routing Algorithm

Incorporating multiple unload locations will have minimal impact on the routing algorithm structure and will not change the trade-offs already performed. If hopping is available for a connector, the cargo selection algorithm will be replaced, adding to the total travel time in addition to outputting the cargo to load. The total travel time will be based on the predicted queue to the first beach location and the total trip time to complete the hops and unload times at selected locations. The option to visit multiple unload locations incorporated a traveling salesman problem into the dynamic loading algorithm and can be incorporated without changing the process for the dynamic routing algorithm.

CHAPTER X

DETAILED MODEL DESCRIPTION

This chapter details the application of the concepts that have been developed thus far into an operating model of the Sea Base. The description demonstrates the application of DES to the Sea Base. Following the general considerations of using DES, a detailed description of the incorporation of the scenario, loading, and routing problems steps through the matrix formulation.

10.1 Notes about Modeling using DES

To formulate a model as a discrete event simulation, each agent must go through a process of waiting and queuing. Each waiting step in the process takes a certain amount of time, determining when the next event will occur. For example, when the MEC is traveling from the Sea Base to shore, it must cover a certain distance and the speed is known so time between the events of leaving the Sea Base and arriving at shore is calculated from the known information and no additional calculations are made for that MEC during that time. For queues, when an event occurs, the queue is reexamined to see if the object can be moved forward. For example, vessels queue waiting for an open beach landing spot and when a vessel leaves a landing spot, it triggers the next vessel to move to the next step of beaching and unloading.

SimPy was selected as the framework for constructing this simulation because it is free and open source. The python based construct allows for the incorporation of additional modules, such as NumPy and LPSolve55, used in this model for matrix manipulation. There are no limits in the size of imported or exported data through the use of comma delimited files. SimPy is an object oriented, text based framework that enables object oriented programming and the development of common scripts as

modules that can be used for different types of objects.

10.2 Object Properties

The connector, supply vessels, resupply points, and beaches are initialized as individual objects and undergo a number of common processes as was discussed in the previous chapter. These processes were abstracted in the model so a single section of code could be used for all types of objects. To use these general scripts, each object must have a common set of attributes defined. Individual characteristics of the objects are stored as properties of the object that are continually updated in a global matrix. The spots vector tracks the types of interfaces that the object can use and has available while the wait vector tracks the time required to load or unload using that interface. The cargo on board or ashore is tracked as a vector, using the generic cargo vector as a base. The cargo wanted vector is the cargo a vessel needs for its next mission or cargo desired by an on-shore location. The cargo to be carried is determined by a subroutine using the lift capability and available area as well as the efficiencies. A cargo capability vector identifies what types of cargo can not be carried on an object.

The location is where the object starts and the goal is the endpoint of the mission. Each object has a type, for example, a resupply point, beachhead, and cargo ships when stationary are 'cargo' objects that supply cargo while a 'connector' object transfer cargo between cargo points. It is probable, but not necessary for a vessel to remain as a single type of object throughout the simulation. For example, a cargo ship leaving a port would be a connector but once it is on station at the Sea Base, it would be a cargo object. When an object is in motion, it must have a purpose, either to pull cargo to meet a cargo desired or to push its cargo on board to a destination.

The fuel usage of an object must be included for different levels of effort such as full power and idling. More detailed fuel equations can be incorporated, but currently

an average or best guess fuel usage is an input to the analysis. Repair requirements are modeled as a necessary wait after the mean time between failures is exceeded and the object reaches a location where repair can be completed. This requires the input of a mean time between failures and a mean time to repair.

For some types of objects, these properties are not applicable but a placeholder must be used since these properties will be used to develop matrices and the placeholders assure alignment of rows. Matrices are compiled for location, spots, cargo have, cargo want and type and as these variables are edited as individual properties of the object, they must also be updated in the overall matrices.

10.3 Generic Process

The most generic process used in this model is for a basic connector. This process is:

1. Identify mission to set the cargo load point and unloading point- this will be described as a separate procedure
2. If needed, travel to load point
3. Identify cargo vessels or objects at load location
4. Identify cargo to load
5. Down select cargo vessels based on cargo available
6. Select quickest interface to load cargo on down selected cargo vessels, remove this interface and incompatible interface from those available
7. Remove cargo from cargo available on cargo vessel
8. Wait loading time and cargo is added to connector
9. Free up used and incompatible interfaces
10. Travel to unloading point

11. Identify cargo objects at unload location
12. Down select based on which object needs the cargo on the connector
13. From these cargo objects, select quickest interface
14. Remove this interface and incompatible ones
15. Wait time to unload
16. Cargo is now available at unload location
17. Release interface and incompatible spots
18. Identify next mission

The process to identify the mission is listed below and summarized in Figure 40:

1. For each load (pull) and unload (push) location, repeat steps 2 through 8
2. Identify cargo needed at the push location
3. Calculate what can be carried on connector and is available at pull location
4. Identify possible spots to use based on cargo objects at location - this calculation does not use the current spots, but all spots
5. Identify the fastest predicted wait from the possible spots, add loading time for this connector to the selected spots predicted wait
6. Calculate expected wait time at pull location using predicted waits at possible spots
7. Calculate travel time for current location to pull location then push location
8. Calculate total trip time, travel, waiting, and load/unloading where travel time is subtracted from wait time

9. Identify best pull/push pair which define the next mission

10. Update global predicted wait matrices based on selected push/pull pair

10.4 Abstracted Processes

Whenever possible within the model, the concepts were abstracted to a more generalized script allowing the reuse of large sections of the code by every object. These code section are modular and represent one step in the calculations, so the section can be rearranged as needed for a class of objects. A brief overview of the scripts are provided here and an example execution of the mathematical construct follows.

Given a location and a goal, the distance calculating script calculates the distance to be traveled by the object. This uses a look-up table provided by the user. Additional locations and goals may be added but must be included in the distance look-up table.

Once a connector object arrives at a location, it will need to find the cargo to load (if loading), a cargo object that has the cargo it needs or that needs the cargo the connector is carrying. This process was broken down into several separate scripts so the scripts can be used individually or as a complete process, depending on the information needed. The first section performs two comparisons, the first finds the objects, which include other vessels and beach landing zones, with locations that match the goal of the connector. The second finds which of those objects are cargo sources/sinks. If the connector is looking to load cargo, it must select the cargo to load. This is based off a prioritized list of cargo that needs transport and the lift and area inputs, along with the efficiencies of the connector.

The next section finds the objects that have the desired cargo or want the cargo carried. If the connector is a "pull" then the compiled "cargo have" matrix is compared row by row to the vector of "cargo want" for the connector. In the case of "push", the connectors "cargo have" is compared to the compiled "cargo want". A

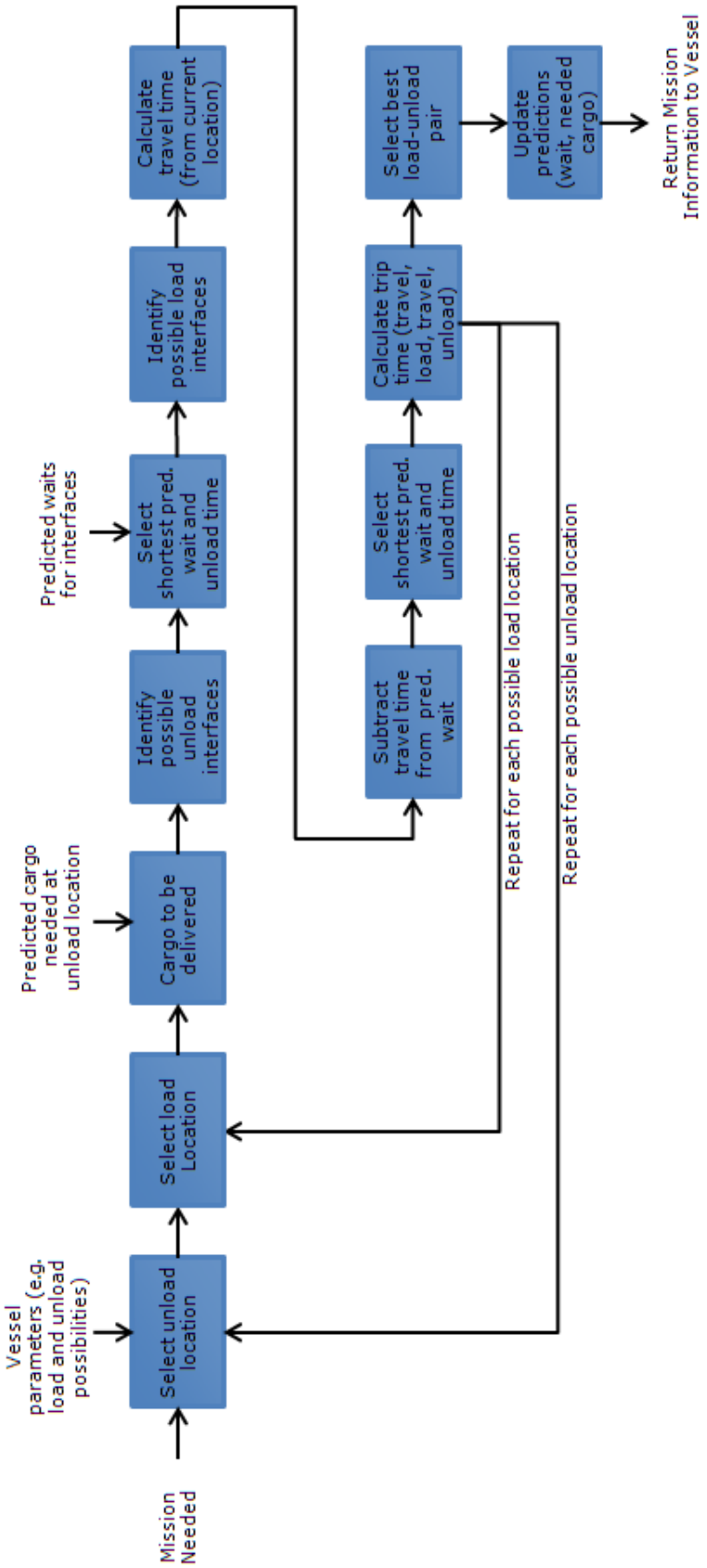


Figure 40: Mission Selection Process

vector is developed that identifies which objects match the desired cargo needs of the connector. It is also necessary to see which objects have interface options available that are usable by the connector. The cargo and interface compatibility are combined with the location and type vectors previously calculated and are applied to the combined spots matrix. This yields a matrix which identifies only the objects that have the correct object type, location, and cargo as well as compatible interface capabilities.

The next step is to identify which of these identified objects and interface types would be the quickest. The wait vector for the connector is applied and the minimum interface time selected. The script has now identified the object that the connector can interface with, the type of interface to be used and the amount of time it will require. The value in the compiled spots matrix for this spot is changed to zero and is no longer available for another asset. The different interface methods on a single object may not be compatible as they may require the use of the same physical equipment or manpower. The next step is to identify these incompatible interfaces and remove those options from the compiled spots matrix as well. The interface type and the incompatible interfaces are stored and are restored in the spots matrix once the loading or unloading is completed. The connectors "cargo want" and "cargo have" variables are updated to reflect the action as well as the rows for the connector and the cargo object in the compiled cargo have and cargo want matrices.

The next step is to calculate the queuing, loading/unloading and travel times. A global matrix is used to track the queuing times based on which connectors are waiting to use each spot. This includes connectors waiting at, and traveling to, the spot. The loading/unloading times are given by the initial input for each connector type. These queuing and loading/unloading times are summed to create a resultant matrix. Then the spot with the shortest time is extracted. Next, the travel time is calculated using the distance to the destination and the speed of the calculation. All

of the times are summed together while keeping in consideration the fact that the queuing time will be reduced over the course of travel. Lastly, these calculations are repeated for every possible pair of push and pull locations and the pair associated with the shortest total time is extracted. The connector's new goal is then set to the pull location. The corresponding push location is recorded and will become the connector's goal after it has picked up cargo.

In order for the dynamic goal setting function to work, two global matrices called Predicted waits and Predicted cargo are used to keep track of predicted queuing time and cargo use, respectively. After a connector selects a new goal, it decides which interface it will use and the associated loading/unloading time. A matrix called wait time is created to hold this predicted loading/unloading time. Next, if the connector's mission is "pull", it decides what cargo it wants using the previously defined choose cargo function. A matrix called predicted cargo is created to hold this predicted cargo use. Then, the wait time and predicted cargo matrices are added to the global matrices, Predicted waits and Predicted cargo, respectively. These global matrices are available for use by all connectors when they decide on a new goal. Finally, when a connector is done loading/unloading, it removes its wait time and predicted cargo matrices from the respective global matrices.

10.5 Example Mathematical Construct

A connector has arrived at the Sea Base and wants to load for its next trip to shore, then travels to shore.

$$\text{Name Matrix} = \begin{pmatrix} ISB \\ Cargo \\ Connector \\ Beach \end{pmatrix}$$

Connection Options: Cargo ship (side and rear) and connector landing on beach

Step 1: Identify cargo objects at Sea Base

$$\underset{\text{Location}}{\begin{pmatrix} ISB \\ SeaBase \\ SeaBase \\ Beach \end{pmatrix}} == SeaBase \rightarrow \underset{\text{Location Vector}}{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}}$$

$$\underset{\text{Type}}{\begin{pmatrix} Cargo \\ Cargo \\ Connector \\ Cargo \end{pmatrix}} == Cargo \rightarrow \underset{\text{Type Vector}}{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}}$$

$$Min \begin{pmatrix} \text{Location Vector} & \text{Type Vector} \\ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, & \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \end{pmatrix} = \underset{\text{Cargo at Location}}{\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}}$$

Step 2: Identify cargo that needs to be brought to shore and will fit on connector. The connector is looking to load cargo so its mission is to "pull" cargo from cargo objects.

Cargo Schedule: [10 0 30 40] Need to deliver 10 of cargo category 0, 30 of category 2 and 40 of category 4.

Cargo Priority: [1 1 2 2] Categories 0 and 1 should be delivered before 2 and 3.

Weight vector:[10 15 20 25] (in LT) and Area vector: [50 100 120 80] (in sqft)

Compatibility Vector: [1 1 1 1], so the connector can carry any type of cargo that will fit aboard. Implemented by multiplying the cargo schedule element by element with the compatibility vector to form the maximum number of that type of cargo carried

on board (Needed vector in LP formulation).

Linear Program formulation: New Priority (P_i):

$$\frac{\frac{Cargoweight_i}{Avgweight} + \frac{Cargoarea_i}{Avgarea} + 1}{Priority_i}$$

$$Maximize \sum_i P_i x_i$$

subject to: subject to:

$$\sum_i weight_i x_i \leq maxlift$$

$$\sum_i area_i x_i \leq maxarea$$

$$x_i - \sum_j cargo_{j,i} * y_j \leq 0 \text{ for all i :}$$

$$\sum_j y_j = 1$$

$$x_i \geq 0 \text{ for all i}$$

$$x_i \leq needed_i \text{ for all i}$$

For max lift of 400 LT and max area of 5000 sqft, the solution of this LP gives a loadout of 10 from category 0 and 15 from category 2 which is 400 LT and 2300 sqft. This load-out is weight limited.

Note: This formulation is for a connector without the option for multiple unload locations. For that case, this LP formulation would be replaced as described in section 9.7.2.

Step 3: Find a cargo object that has needed cargo and available interface usable by MEC

$$\underset{\text{Cargo Have}}{\begin{pmatrix} 20 & 8 & 24 & 10 \\ 24 & 6 & 20 & 5 \\ 0 & 0 & 0 & 0 \\ 12 & 4 & 30 & 10 \end{pmatrix}} \geq \underset{\text{Cargo want}}{\begin{pmatrix} 10 & 0 & 15 & 0 \end{pmatrix}} \rightarrow \underset{\text{Needed Cargo}}{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}}$$

$$Min\left(\begin{matrix}\text{Needed Cargo}\\ \begin{pmatrix}1\\1\\0\\1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Cargo at Location}\\ \begin{pmatrix}0\\1\\0\\0\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\1\\0\\0\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Connector Spots}\\ \begin{pmatrix}1&1&0&1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\1\\0\\0\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Available Spots}\\ \begin{pmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\end{matrix}$$

Note: If no cargo object exists with the needed cargo and an available interface, the connector will queue and restart at Step 1.

Step 4: Select quickest interface, remove it and incompatible interfaces from those available

$$\begin{matrix}\text{Available Spots}\\ \begin{pmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\end{matrix} * \begin{matrix}\text{Wait Times}\\ \begin{pmatrix}10&60&0&45\end{pmatrix}\end{matrix} = \begin{matrix}\text{Wait for Spots}\\ \begin{pmatrix}0&0&0&0\\0&60&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\end{matrix}$$

The connector will take 60 min to load, using the side connection of the cargo ship. Incompatible spots are spots that can be used at the same time for load and/or unloading. When one of these spots are selected, it removes the incompatible spots

as well.

$$\overset{\text{Incompatible}}{\begin{pmatrix} & pier & side & rear & beach \\ pier & 1 & 0 & 0 & 0 \\ side & 0 & 1 & 1 & 0 \\ rear & 0 & 1 & 1 & 0 \\ beach & 0 & 0 & 0 & 0 \end{pmatrix}} * \overset{\text{Selected Spot}}{\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}} = \overset{\text{Spots to Remove}}{\begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}}$$

$$\overset{\text{Spots}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}} - \overset{\text{Spots to Remove}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} = \overset{\text{Spots}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}}$$

Step 5: Reserve cargo on cargo object and load cargo

At start of loading, cargo is no longer available on cargo ship:

$$\overset{\text{Cargo Have}}{\begin{pmatrix} 20 & 8 & 24 & 10 \\ 24 & 6 & 20 & 5 \\ 0 & 0 & 0 & 0 \\ 12 & 4 & 30 & 10 \end{pmatrix}} - \overset{\text{Cargo want}}{\begin{pmatrix} 10 & 0 & 15 & 0 \end{pmatrix}} = \overset{\text{Cargo Have}}{\begin{pmatrix} 20 & 8 & 24 & 10 \\ 14 & 6 & 5 & 5 \\ 0 & 0 & 0 & 0 \\ 12 & 4 & 30 & 10 \end{pmatrix}}$$

After waiting the loading time (60 min in this example), the cargo is now on board the connector

$$\overset{\text{Cargo want}}{\begin{pmatrix} 10 & 0 & 15 & 0 \end{pmatrix}} \rightarrow \overset{\text{Cargo want}}{\begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}} \textit{ and } \overset{\text{Cargo Have}}{\begin{pmatrix} 20 & 8 & 24 & 10 \\ 14 & 6 & 5 & 5 \\ 10 & 0 & 15 & 0 \\ 12 & 4 & 30 & 10 \end{pmatrix}}$$

Step 6: Free up interface option(s)

$$\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&0&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}+\begin{matrix}\text{Spots Released}\end{matrix}\begin{pmatrix}0&0&0&0\\0&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix}=\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}$$

Step 7: Change mission and travel to shore. At this point, the connector's goal becomes to push cargo to a cargo object. In this example, the location of the goal is Beach. The connector must travel from its current location, Sea Base, to Beach by looking up the distance between the locations.

$$\begin{matrix}\text{Distances}\end{matrix}\begin{pmatrix}&ISB&SeaBase&Beach\\ISB&0&200&300\\SeaBase&200&0&100\\Beach&300&100&0\end{pmatrix}$$

Step 8: (Step 1 for beach) Identify cargo object at beach

$$Min\begin{matrix}\text{Location Vector}\end{matrix}\begin{pmatrix}\begin{pmatrix}0\\0\\1\\1\end{pmatrix},&\begin{matrix}\text{Type Vector}\end{matrix}\begin{pmatrix}1\\1\\0\\1\end{pmatrix}\end{pmatrix}=\begin{matrix}\text{Cargo at Location}\end{matrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Step 9: (Step 3 for push) Find a cargo object that needs cargo on connector and has an available interface usable by connector

$$\begin{matrix}\text{Cargo Want}\end{matrix}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\10&0&30&40\end{pmatrix}>=\begin{matrix}\text{Cargo have}\end{matrix}\begin{pmatrix}10&0&15&0\end{pmatrix}\rightarrow\begin{matrix}\text{Desire Cargo}\end{matrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

$$Min\left(\begin{matrix}\text{Desire Cargo}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Cargo at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Connector Spots}\\ \begin{pmatrix}1&1&0&1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Available Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}\end{matrix}$$

Step 10: (Step 4) Select quickest interface, remove it and incompatible interfaces from those available

$$\begin{matrix}\text{Available Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}\end{matrix} * \begin{matrix}\text{Wait Times}\\ \begin{pmatrix}10&60&0&45\end{pmatrix}\end{matrix} = \begin{matrix}\text{Wait for Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&45\end{pmatrix}\end{matrix}$$

The connector will be unloaded at the beach in 45 min.

Incompatible interfaces are determined as seen above, for this case there are no incompatible interfaces for the beach.

$$\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}-\begin{matrix}\text{Spots to Remove}\end{matrix}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}=\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&0\end{pmatrix}$$

Step 11: (Step 5 in reverse) Unload cargo

After the cargo is unloaded, there is no cargo on the connector and the cargo is now on the beach

$$\begin{matrix}\text{Cargo Have}\end{matrix}\begin{pmatrix}20&8&24&10\\4&6&5&5\\10&0&15&0\\12&4&30&10\end{pmatrix}+\begin{matrix}\text{Cargo have}\end{matrix}\begin{pmatrix}10&0&15&0\end{pmatrix}=\begin{matrix}\text{Cargo Have}\end{matrix}\begin{pmatrix}20&8&24&10\\4&6&5&5\\10&0&15&0\\22&4&45&10\end{pmatrix}$$

$$\begin{matrix}\text{Cargo have}\end{matrix}\begin{pmatrix}10&0&15&0\end{pmatrix}\rightarrow\begin{matrix}\text{Cargo have}\end{matrix}\begin{pmatrix}0&0&0&0\end{pmatrix}$$

Step 12: (Step 6) Free up interface option(s)

$$\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&0\end{pmatrix}+\begin{matrix}\text{Spots Released}\end{matrix}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}=\begin{matrix}\text{Spots}\end{matrix}\begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}$$

Note: If multiple unload points are possible, the connector would change its goal to its next beach option, such as an austere port, and repeat steps 7 - 12.

Step 13: Change mission

The connector has completed it "push" mission and now much be given a new mission

and goal. The connector will choose a pair of pull and push locations.

Possible pull locations: Sea Base and Intermediate Staging Base

Possible push locations: Beach

Step 14: Select compatible pull locations for the Beach

Both the Sea Base and Intermediate Staging Base are compatible pull locations to push at the Beach.

Step 15: : Calculate expected wait times at Beach Identify cargo that needs to be brought to Beach and will fit on connector. (Step 2)

$$\overset{\text{Cargo want}}{\begin{pmatrix} 0 & 0 & 15 & 5 \end{pmatrix}}$$

Identify cargo object at Beach (Step 8)

$$Min \begin{pmatrix} \overset{\text{Location Vector}}{\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}} , \overset{\text{Type Vector}}{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}} \end{pmatrix} = \overset{\text{Cargo at Location}}{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}}$$

Find a cargo object that needs the cargo that is expected to be on the connector and has an available interface usable by connector. The interface is not restricted by current availability. (Step 9)

$$\overset{\text{Cargo Want}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 40 \end{pmatrix}} >= \overset{\text{Cargo want}}{\begin{pmatrix} 0 & 0 & 15 & 5 \end{pmatrix}} \rightarrow \overset{\text{Desire Cargo}}{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}}$$

$$Min\left(\begin{matrix}\text{Desire Cargo}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Cargo at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{All Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&1&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Connector Spots}\\ \begin{pmatrix}1&1&0&1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}$$

$$Min\left(\begin{matrix}\text{Usable Spots}\\ \begin{pmatrix}1&0&0&0\\0&1&0&0\\1&1&0&1\\0&0&0&1\end{pmatrix}\end{matrix}, \begin{matrix}\text{Needed at Location}\\ \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\end{matrix}\right) = \begin{matrix}\text{Available Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}\end{matrix}$$

Note: "Cargo want" is used instead of "Cargo have" since it is the cargo that the connector is expected to have. Additionally, "All Spots" is used instead of "Spots" in order to account for the interfaces that are currently occupied.

Find the expected wait for all possible interfaces. First the expected loading times are calculated (Step 10) and then the expected queuing times are added. In this example, there are two similar connectors queued at the beach.

$$\begin{matrix}\text{Possible Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}\end{matrix} * \begin{matrix}\text{Wait Times}\\ \begin{pmatrix}10&60&0&45\end{pmatrix}\end{matrix} = \begin{matrix}\text{Loading Time for Spots}\\ \begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&45\end{pmatrix}\end{matrix}$$

$$\underset{\text{Possible Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}} * \underset{\text{Predicted Waits}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&45\end{pmatrix}} = \underset{\text{Predicted Wait for Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&90\end{pmatrix}}$$

$$\underset{\text{Loading Time for Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&45\end{pmatrix}} + \underset{\text{Predicted Wait for Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&90\end{pmatrix}} = \underset{\text{Wait for Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&135\end{pmatrix}}$$

Select the quickest interface. The loading time (45 min) and expected wait (90 min) are recorded for this interface. The incompatible interfaces are determined and recorded (there are not incompatible interfaces for the beach).

Step 16: Calculate expected wait times at Sea Base (Step 14)

$$\underset{\text{Loading Time for Spots}}{\begin{pmatrix}60&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}} + \underset{\text{Predicted Wait for Spots}}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}} = \underset{\text{Wait for Spots}}{\begin{pmatrix}60&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}}$$

Select the quickest interface. The loading time (60 min) and expected wait (0 min) are recorded for this interface. The incompatible interfaces are determined and recorded as was done in step 4.

Step 17: Calculate total time for Beach, Sea Base pair

Calculate travel time from current location to pull location to push location

Distance from current location (Beach) to pull location (Sea Base) to push location (Beach) = 200 nmi

$$\text{Travel time} = \frac{Distance}{Speed} = \frac{200nmi}{20knts} * 60 = 600min$$

Combine loading time, wait time, and travel time:

$$\text{Total time} = 45min + 600min = 645min$$

In this example, the expected wait time (90 min) is less than the travel time (600 min), so it is not included. This is because the other connectors in the Beach queue will have finished unloading by the time the connector has returned to the Beach.

Step 18: Calculate expected wait times at Intermediate Staging Base (Step 16)

$$\begin{matrix}\text{Loading Time for Spots}\end{matrix} \quad \begin{matrix}\text{Predicted Wait for Spots}\end{matrix} \quad \begin{matrix}\text{Wait for Spots}\end{matrix}$$

$$\begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Select the quickest interface. The loading time (10 min) and expected wait (0 min) are recorded for this interface. The incompatible interfaces are determined and recorded (there are not incompatible interfaces for the pier).

Step 19: Calculate total time for Beach, ISB pair (Step 17)

With a similar method, the total time is calculated to be:

$$\text{Total time} = 10min + \frac{600nmi}{20kts} * 60 = 1810min$$

Step 20: Select the best pull/push pair

$$\text{Goodness of pair} = \frac{\text{total trip time}}{\sum_i P_i x_i}$$

Since the same cargo is available at both locations, the connector will pull at the Sea Base and push at the Beach since the total time is less. If no pair is found, the connector will wait at location and intermittently recheck for a new mission.

Step 21: Update the global matrices

Store predicted loading times at pull and push locations.

$$\underset{\text{Pull Loading Time}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 60 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} * \underset{\text{Spots to remove}}{\begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}} = \underset{\text{Pull Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 60 & 60 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}$$

$$\underset{\text{Push Loading Time}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 45 \end{pmatrix}} * \underset{\text{Spots to remove}}{\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}} = \underset{\text{Push Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 45 \end{pmatrix}}$$

$$\underset{\text{Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 90 \end{pmatrix}} + \underset{\text{Pull Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 60 & 60 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} + \underset{\text{Push Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 45 \end{pmatrix}} = \underset{\text{Predicted Wait}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 60 & 60 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 135 \end{pmatrix}}$$

Store predicted cargo pickup from pull location and add to global matrix.

$$\underset{\text{Predicted Cargo}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} + \underset{\text{Predicted Cargo Usage}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 30 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} = \underset{\text{Predicted Cargo}}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 30 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}$$

Step 22: The goal is changed to the chosen pull location and the push location is stored as the following goal (used in Step 7). The entire process is repeated.

10.6 Processes for Different Types of Vessels

When generating DES in SimPy, it is important to note that calculation may be completed in common scripts, such as those described above, but all queues and

waits must be executed in the object's process. For example, the minimum wait time to reload can be found using the same script for the MEC and LCAC but the hold statement to model that wait must be in the individual object process. The process for connector objects will resemble the example process detailed above and all the mathematical constructs remain the same, although some may be removed and the order changed as determined by the connector type. Since many of the vessels/ vehicles go through similar processes, a hierarchy was created so similar objects inherit common properties and processes.

Connector

- Supply Ship (Surface)
  - Medium Exploratory Connector (MEC)
  - Joint High Speed Vessel (JHSV)
  - Dry Cargo/Ammunition Ships (T-AKE)
  - Logistics Support Vessel(LSV)
  - Landing Craft Utility 2000 class(LCU-2000)
- MV-22 Osprey (Aerial)
- Carried to Theater Connector (Organic, Surface)
  - Improved Navy Lighterage System (INLS)
  - Landing Craft Utility 1600 class (LCU-1600)
- Helicopter (Organic)
  - CH-46 Sea Knight
  - CH-53 Sea Stallion
- Air Cushion Vehicle

- Landing Craft Air Cushion (LCAC)
- Landing Craft Air Cushion - Replacement (LCAC-R)

SB Cargo Ship

- Large, Medium-Speed, Roll-on/roll-off ship (LMSR)
- Landing Helicopter Deck (LHD)
- Landing Platform Dock (LPD)
- Landing Ship, Dock (LSD)

Mobile Landing Platform (MLP)

Shore

- Staging Base
- Port
- Austere Port
- Beach Landing Zones

These vessels and locations are currently implemented in the model. The processes developed and the baseline properties implemented in the model were based on the Amphibious Ships and Landing Craft Data Book [188], open source information on the internet [1], and iteration with subject matter experts from ONR.

10.6.1 Connector

All of these ships serve as connectors moving cargo to meet a demand. First, a mission is selected, establishing locations to pick up and drop of cargo. The connector travels to the pick-up location and identifies a spot on a ship, at a port, or at a supply base to load cargo. After traveling to the push goal, a spot is selected and the cargo

offloaded. At this point, the connector identified its next mission and travels to a cargo supply point to load again. The process to return to the cargo point and reload is the same process as traveling to shore but instead of pushing cargo, it is pulling, so the process is not duplicated twice in the connector class. This process is depicted in Figure 41. The onload and offload locations are properties of the individual ships. These connectors are assumed to start equally distributed between the Intermediate Staging Bases (ISB) with no cargo on board.

There are different types of connectors in involved in this group. The most basic is the surface connector. This differs from the aerial connector because the surface connector can repair as a self contained unit while the aerial connector can only repair once landed. This is a flag within the connector process so it is only necessary to include surface versus aerial as a property of the vessel class. The other property included in the class is if the vessel is organic. In this case, it is not self deployed and must be ferried into theater by another vessel. The loading and unloading is handled on the connector side but must be added to the SB Cargo Ship, described later.

10.6.1.1 Supply ship

MEC, JHSV, TAKE, LSV and LCU-2000

All of the supply ships are surface connectors. These ships have a flag for the ability to unload at multiple locations. The MEC is the only class that must change the general process.

MEC

The MEC process is that of a basic connector. Since the MEC must transition before reaching the shore the travel time algorithm is replaced in this class, but the general process is inherited from the connector class. The distance at which transition occurs is removed from the total travel distance. The time to travel to the transition distance, transition, and travel to shore are calculated using time equals distance

divided by speed.

10.6.1.2 MV-22

The MV-22 is handled the same as an aerial connector, as it is assumed it can queue in the air. If it is determined to act more like a helicopter, where landing is a priority, not matching cargo, then the process can be switched to that of the helicopter class.

10.6.1.3 Connectors Carried into Theater

INLS and LCU-1600

These connectors must be carried into theater on designated connectors. Once the ferrying ship arrives in theater, these connectors are offloaded first. These connectors then become independent vessels that travel between set points. These will include the SB and shore, but may also include a nearby port, determined by the inputted vessel properties.

10.6.1.4 Helicopters

CH-46 and CH-53

Helicopters must be carried aboard another vessel to reach the Sea Base. Once deployed, they follow a slightly modified procedure because they can not queue as an individual asset. This process is depicted in Figure 42. If a landing spot can not be identified that has the cargo the helicopter would like to carry or drop off, depending on the mission, the helicopter will land at the first available spot. This is to minimize the time in the air. Any repairs needed are completed once the helicopter lands. Once the helicopter has landed, it will wait until the cargo becomes available at that source and will not change cargo supply locations to match its cargo needs. Since this process is significantly different from the general connector process, the entire process is overwritten within the connector class.

Table 10: Run Time Comparison - With and without Helicopters

Case	Without Helos (sec)	With 2 Helos (sec)
1	2.3	9.2
2	2.7	3.9
3	14.7	24.2
4	9.6	7.6
5	5.3	11.0
6	8.8	17.9
7	4.6	13.0
8	13.0	30.8

Note: Helicopters are much more computationally intensive than aurface connectors. To demonstrate this, eight cases were run without helicoptors and repeated with one helicopter of each type. Table 10 shows that for most cases that is an increase in run time by introducing helicopters.

10.6.1.5 Air-Cushion Connector

LCAC and LCAC-R/SSC

Air-cushioned vessels are another special case of connector and required the development of a process that overwrites the general connector process for these vessels. This difference is due to the MLP being an option to ferry the ACVs closer to shore if the SB Cargo Ships are outside their range. This process is visualized in Figure 43. The ACVs start at the ISB with no cargo and are transported to the Sea Base aboard another vessel, such as the MLP. If the Sea Base is positioned within the range of the ACV then the ACV serves as a basic connector, following the general procedure for a connector. It travels to its goal and then must queue for a loading spot with the correct available cargo or an unloading spot that requires the cargo onboard. It would then load or unload and move on to its next mission. If the Sea Base is positioned beyond the range of the ACV, the MLP must be used to ferry the ACVs to some standoff distance. Instead of calculating the distance to travel and waiting that amount of time, the ACVs must wait for an MLP to carry them, forming a queue.

Once the MLP reaches the standoff distance, it offloads and the ACVs travel the remaining distance to shore and find a landing zone on the beach. Once unloaded, the ACVs repeat the same process with the ACVs queuing to be ferried by the MLP and unloaded upon arrival at the Sea Base, joining the loading queue.

10.6.2 Sea Base Cargo Ships

LMSR, LHD, LPD, LSD

Cargo ships, including the LMSR, act as cargo objects that serve to interface with connectors and hold cargo. They must start at some initial starting location and travel to the Sea Base or other cargo spot, such as a nearby port. While a cargo object is stationary, no process definition is needed beyond the definition of the initial properties. The type of object can be changed to connector, and movement processes defined, if the cargo object needs to move from one location to another.

These ships can be used to carry helicopters and smaller vessels in its well deck. The ship ferries the helicopters and smaller vessels, such as a LCU-1600 and INLS to the SB, where they become individual connectors and the cargo ship serves as a resupply point. The number of each typeof helicopter and smaller shisp that can be carried are set as vessel properties, but only a single type is selected.

10.6.3 Mobile Landing Platform

The MLP is a special case as it serves as a cargo transfer enabler as well as having the potential to disconnect from the LMSR and travel as an independent object [25]. In the case where the MLP stays attached to the LMSR and does not ferry LCACs, the MLP simply serves to enable a loading connection on the LMSR. The MLP starts at the ISB and ferries connectors as needed. These connectors are released when the MLP reached the Sea Base and full connectors loaded, if available. Otherwise, the MLP identifies an available LMSR and the connection made. Once the MLP is attached, the through MLP loading interface is enabled in the LMSRs properties. In

this way, the MLP becomes a part of the LMSR and is not considered a separate object. When the MLP is used to ferry LCACs or LCACRs, it is assumed to starts unattached and it immediately looks for connectors to load from the queue of loaded objects at the Sea Base. Once the connectors are on board, the MLP finds the distance to shore, less its standoff distance, and travels for that amount of time. Once it arrives, it triggers the reactivation of the connectors so they can travel to shore and unload. The MLP waits at the standoff distance for the return of the connectors, and once they are on board, returns to the Sea Base. Once again, it triggers the reactivation of the connector process when it releases them at the Sea Base. The process repeats with the MLP looking for loaded LCACs or LCACRs and if not enough are present, identifying an LMSR to connect with. This process is summarized in Figure 44.

10.6.4 Shore

Beach Landing Zone and Austere Port

Beaches are established as a stationary object by defining the initial properties and no additional process definition is needed. The different types of landing zones are identified by the interfaces available and those must match the designated connector type.

Staging Base

The intermediary staging base (ISB) is a base where the assets are pre-positioned and a point of resupply. These bases are at a set distance from the theater with an inputted amount of cargo available. These bases can have up to three piers for resupply.

Port

This represents a port near the theater of interest that can be used as a throughput point. This port does not have any cargo at the start of the simulation but serves as a pier where cargo can be offloaded from larger ships that are either faster to offload

at a pier or can not transfer cargo at sea. This cargo is then loaded onto smaller connectors that transfer it directly to the beach.

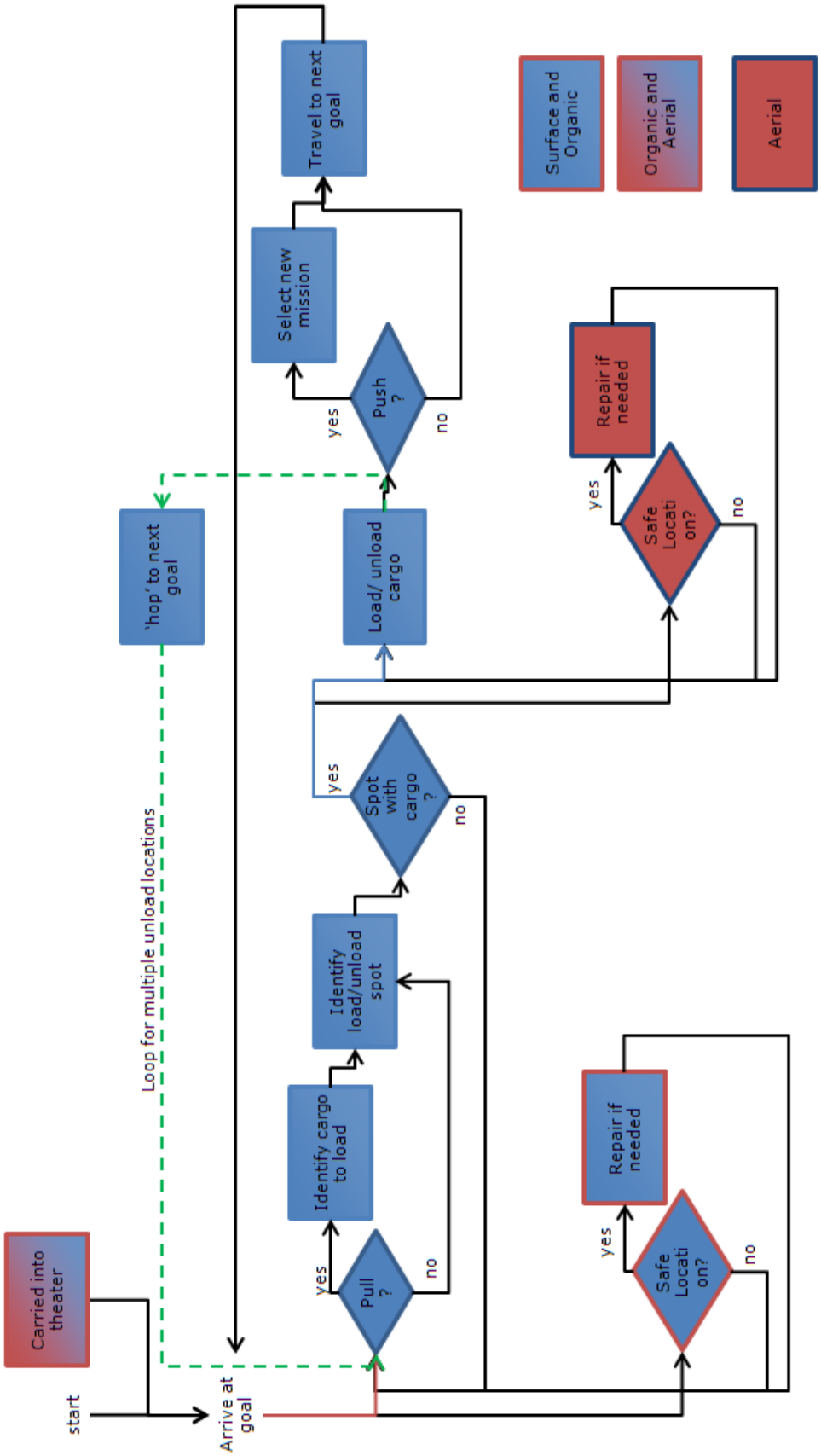


Figure 41: Connector Process

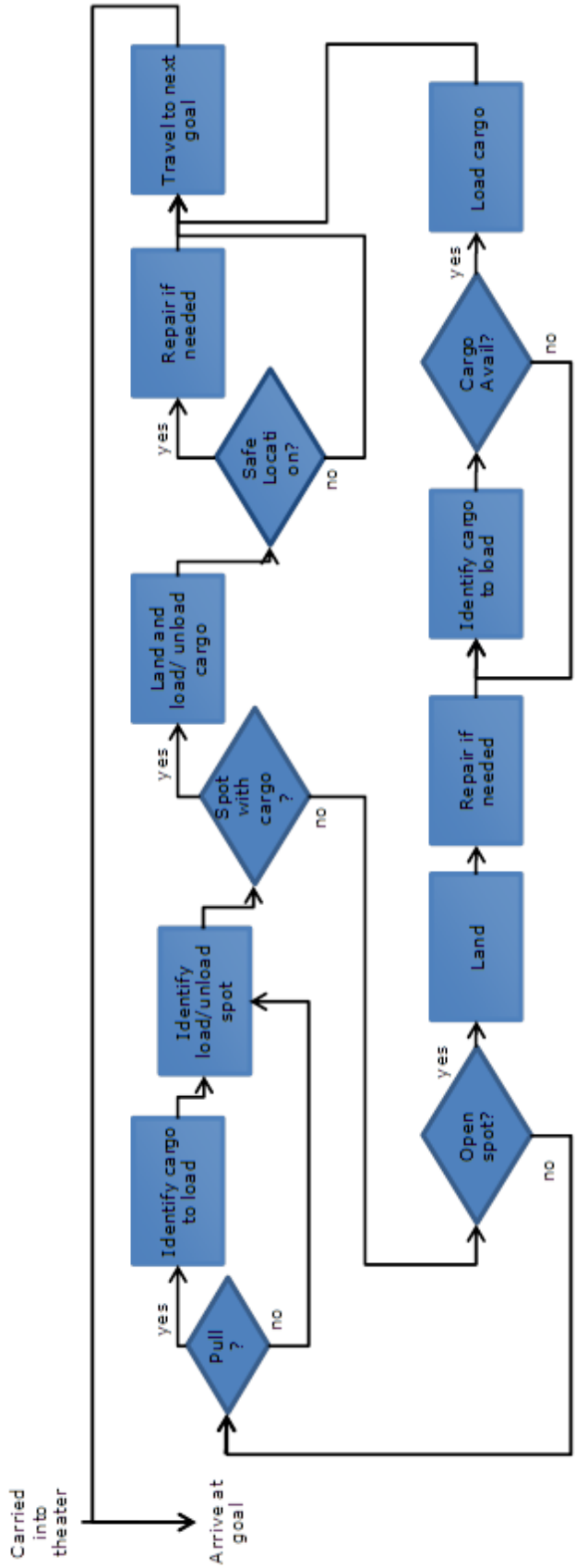


Figure 42: Helicopter Process

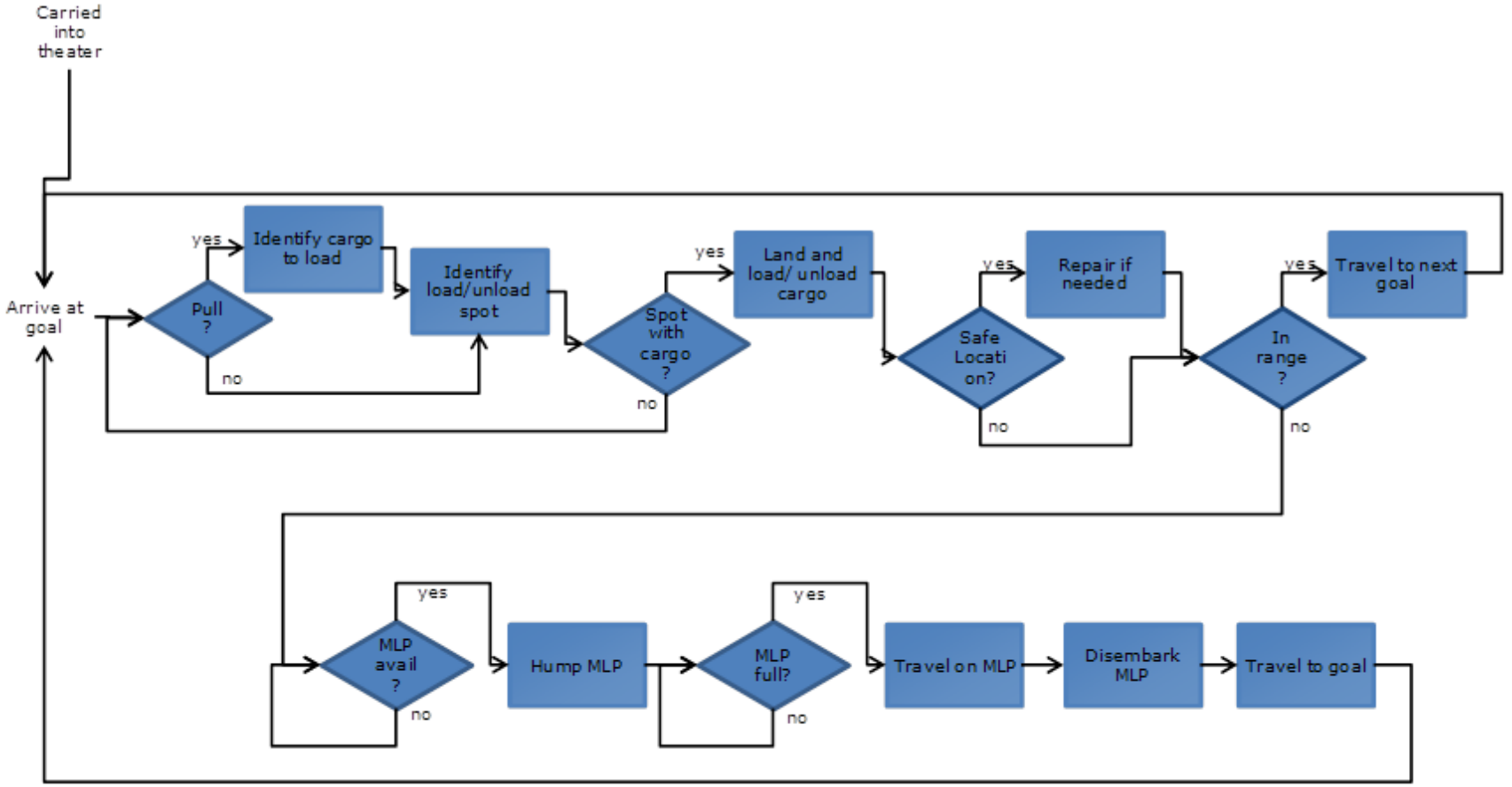


Figure 43: Air Cushion Vehicle Process

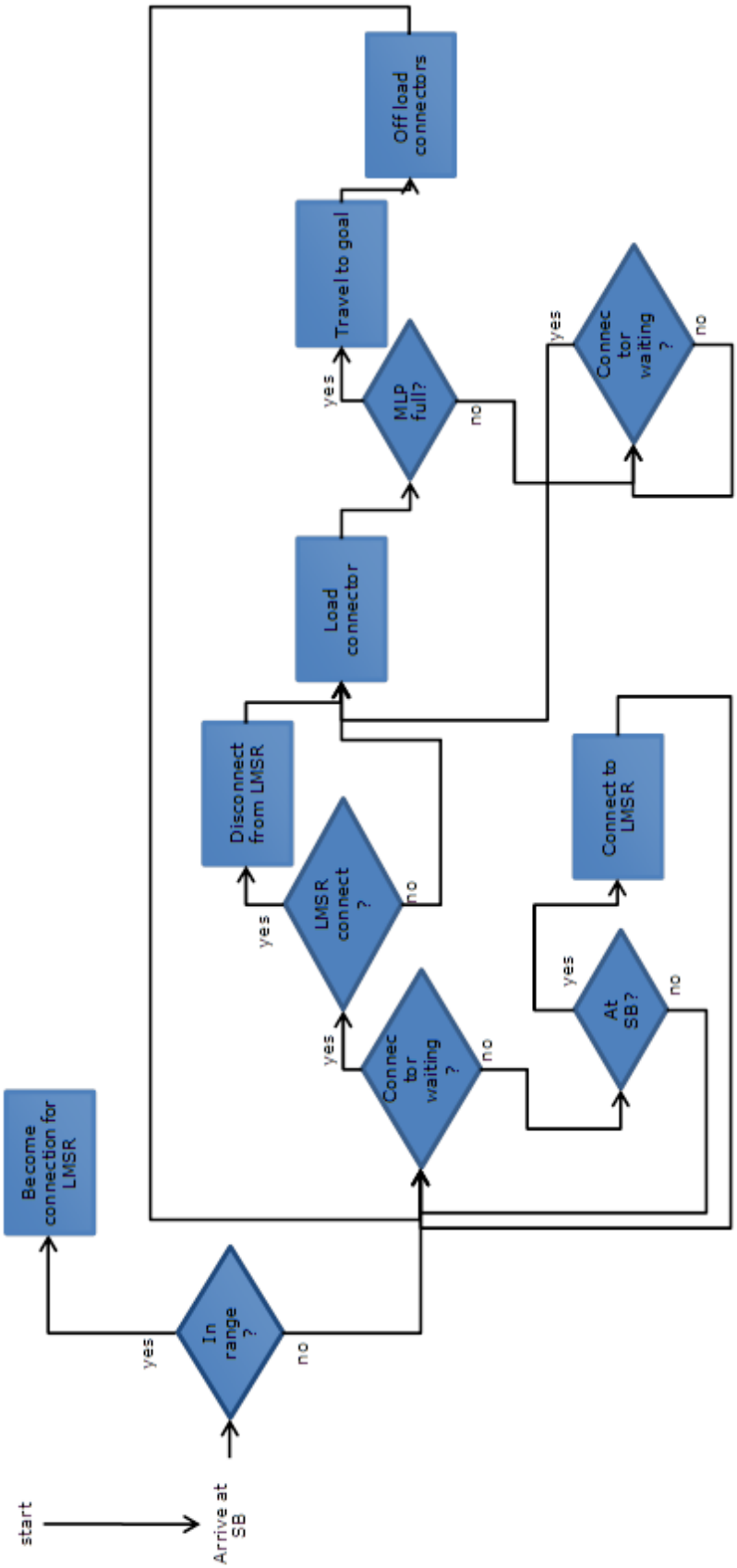


Figure 44: MLP Process

CHAPTER XI

COMPARISON TO EXISTING MODEL

This chapter details the comparison of results generated by CDM Inc. and provided by Ms Kelly Cooper. The scenarios and vessels modeled by CDM Inc. will be repeated using the model developed in this thesis, called the DELAS (Discrete Event Logistics Advanced Simulation) model, with a comparison of assumptions and description of discrepancies. The work by CDM Inc. was based on the T-Craft, but the conceptual connector (MEC) explored in this thesis is flexible enough to capture the concept of the T-Craft. For this chapter, the conceptual connector will be label T-Craft to ease comparison between the models.

11.1 Description of CDM Work

CDM completed a study of six T-Craft concepts and compared their performance to that of a SSC. Three loading options were presenting for the T-Craft options. These options are listed below. In Figure 45, two of these load-outs are presented for the same T-Craft concept. The T-Craft will carry three movers, type determined by cargo type selected while the SSC will carry one mover.

1. Medium Tactical Vehicle Replacements (MTVRs)
    - MK23 Standard MTVRs each carrying up to 6 pallets
2. Logistics Vehicle Systems (LVSs) with Containers
    - MK48 LVS Front Power Unit with MK18 Rear Body Units each carrying 1 8820 ISO container
3. Tractor-Trailers with Containers

- M931 Tractor Truck with M1076 Trailers each carrying 1 8820 ISO container

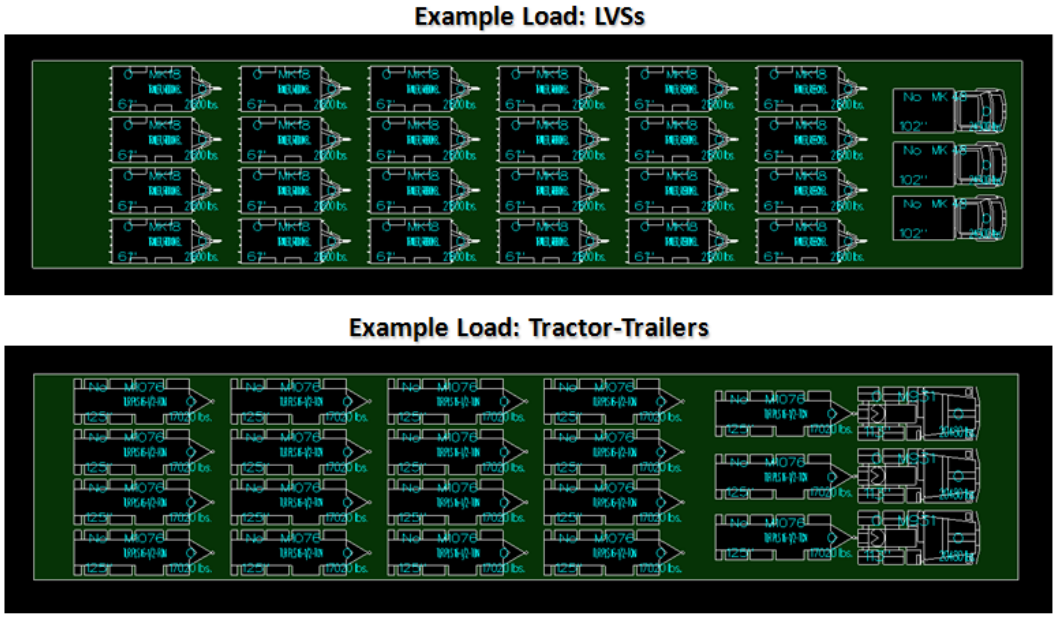


Figure 45: Example Load-Outs

11.1.1 Scenarios

The focus of this study was humanitarian missions to demonstrate the potential use of the T-Craft as a humanitarian connector. Two scenarios were analyzed, a CONUS scenario (e.g., Gulf Coast) and an international scenario (e.g., Haiti). For the CONUS scenario, the Sea Base is 25 to 30 nmi from the four population centers, which are 5 to 10 nmi apart along the coast. Each population center has 25 thousand refugees each receiving two MREs and 1.5 gallons of drinking water per day. Tents are provided for ten percent of the refugees. The operation is two days, with the first delivering two days of supply and the shelter. The follow-on day is one day of supply and an engineer support company personnel and construction materials. For the international scenario. the Sea base is 25 to 40 nmi from five population centers

10 nmi apart on the coast. Each population center has ten thousand refugees, with each refugee receiving 460 g maize and 80 g beans per day and 1.5 gpd drinking water. Shelter is needed for 25 percent of the refugees. The first operational day is to deliver two days of supply and the shelter with the follow-on day of one day of supply and an engineer support company personnel and construction materials. Both scenarios are assumed to include three LMSRS and three T-Craft of the same variant and this is compared to a fleet of nine SSCs. LVS and tractor-trailer loading will be considered, but the MTVR is not considered for the humanitarian scenarios.

11.1.2 Timing Assumptions

To model the cargo delivery process, it is necessary to make several timing assumptions. These timing assumption were made based on LCAC planning factors. The timing assumptions are detailed before and the variant depending timings are given in Table 11.

- Enter critical landing zone = 20 minutes
- Connect to LMSR = 20 minutes
- Load Time = Depends on variant and cargo type
  - MTVRs: Number of Rows × 30 mins
  - LVSs: (Number of Trailer Rows × 40 mins) + 30 mins for the prime movers
  - Tractor-Trailers: (Number of Trailer Rows × 40 mins) + 30 mins for the prime movers
  - Rows logic assumes loading and securing a row can be near-simultaneous
  - 5 percent Penalty to Load Time Due to complexities of the side port ramp, turntable, and narrow form factor of the payload area
- Offload Time = $^1/_3$ Load Time

Table 11: Timing Assumptions

		Option 1	Option 2	Option 3	Option 4	Option 5	Option 6	SSC
	Loading location	Side	Side	Stern	Stern	Stern	Stern	MLP (side)
MTVR	Side load time	245	320	0	0	0	0	70
	Stern load time	0	0	170	170	200	260	0
	Beach unload time	80	100	70	70	80	100	30
LVS	Side load time	408	558	0	0	0	0	70
	Stern load time	0	0	250	250	290	320	0
	Beach unload time	123	163	97	97	110	120	30
Tractor Trailer	Side load time	308	458	0	0	0	0	70
	Stern load time	0	0	210	190	250	330	0
	Beach unload time	97	157	83	83	97	123	30
	Cruising Speed(kts)	41	41	48.3	42	40	40	35
	TC Speed ACV (kts)	5	5	5	5	5	5	-
	Transition time (min)	25	25	20	20	10	10	-
	Payload capacity (LT)	305	700	352	639	300	750	74
	Payload Area (sq ft)	7500	12380	7327	7327	6560	13395	2500

11.1.3 Loading Assumptions

Loading and unload times are based on the assumption that a row of movers can be loaded/unloaded simultaneously. For example, the CDM calculations assume that it takes the same amount of time to unload one row of 2 MTVRs as it would to unload one row of 4 MTVRs. This causes results to favor wider designs. A vessel is loaded until the area in the maximum number of containers or the payload capacity for that concept (including movers and trailers) is met. The maximum number of containers is calculated in advance for each concept and each cargo type. Types of cargo can be mixed within a container. Vessel can hop between landing zones to bring a fuller load, but individual containers can not be split between beaches.

11.1.4 Scheduling Assumptions

CDM schedules starts with vessel 1 and schedules until operating time is reached, then moves on to the next vessel until maximum number of vessels is reached. The algorithm requires landing zone 1 is fulfilled before starting landing zone 2 and so on. This can cause queuing at landing zones since each landing zone can unload one vessel at a time so vessel 1 and 2 can be sent to landing zone 1 at the same time, as seen in Figure 46. Each day of operation is considered separations so if a demand is not met within the time period, it expires and is not transported on sequential day. The Sea base was assumed not to limit operations so vessels are loaded immediately upon arrival.

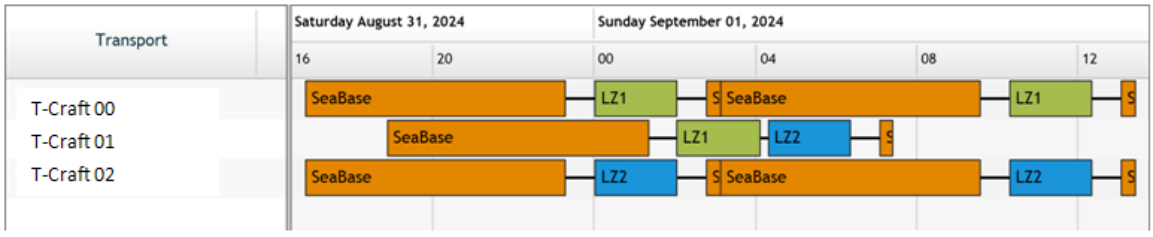


Figure 46: Scheduling Shortcoming

11.2 Comparison of Assumptions

Due to the differences in the formulation of the CDM model and the model developed in this work, there are some differences in assumptions. When considering the loading of the vessels, CDM individually loads the containers on the connector to the exact amount needed on shore. This loading is done outside the discrete event simulation and was not detailed beyond loading diagrams, such as Figure 45.

In contrast, this thesis assumes a container may only contain one type of material and must be transferred as a single unit. The container must be transfered completely full even if only a partial container is needed. A vessel can carry only one type of loading, an LVS system or tractor-trailers, but can carry containers of different types of supplies. The capability of of an 8820 ISO container was calculated using volume for cases of water, cases of MREs, etc. and are given in Table 12. The assumed weight for each of these items is given in Table 13. CDM supplied the assumptions for the engineering company supplies of 9110 lb and 230 ft2 per beach.

In this work, the total weight of a container was calculated using this weight multiplied by the capacity, asses to the tare weight of the trailer and container. The area for the container is based on the footprint of the trailer. This weight and area are used with the dynamic loading algorithm described in Chapter 8 to load the vessels. The weight and area required for the movers is subtracted from the vessels capabilities before the dynamic algorithm is applied. The required number of containers of each type of supply was calculated based on the demand and capacity of the containers, given in Table 14. This requirement was rounded up to the nest whole container because it was assumed containers would not be broken down.

Handling the demand is different between the two models. When simulating for 48 hours, the model for this work assumes the fleet must complete day 1 before going on to day 2. The CDM work treats each day individually. This assumption difference can be worked around and the impact of analyzing the days separately versus together

is examined.

Table 12: Carrying Capacity per Container

	ISO Container
Cases of Water	700
Cases of MREs	800
Tents	65
90 kg Bulk Food	170

Table 13: Assumed Weights

	Weight (lbs)
Case of Water	45
Case of MREs	22
Tent	353
90 kg Bulk Food	198

Table 14: Demand per Population Center

Day	Domestic				International			
	Water	MRE	Bulk	Tents	Water	MRE	Bulk	Tents
Day 1	23	11	0	5	10	0	1	5
Day 2	12	6	0	0	5	0	1	0

CDM schedules ship by ship and must complete one landing zone before moving on to the next. This can cause the queuing seen in Figure 46 and no queuing is considered at the Sea Base. This model considers queuing at the beaches and at the Sea Base. The schedule is based on the most full load that can be sent. This results in the schedule in Figure 47. The dynamic loading algorithm generates load-outs based on the demand at each landing zone and selects based on the greatest priority load. The amount of cargo delivered will be greater because an additional trip is made.

The operational time constraint is dealt with very different where the CDM model does not send a connector if the mission will exceed the operational time limit. This

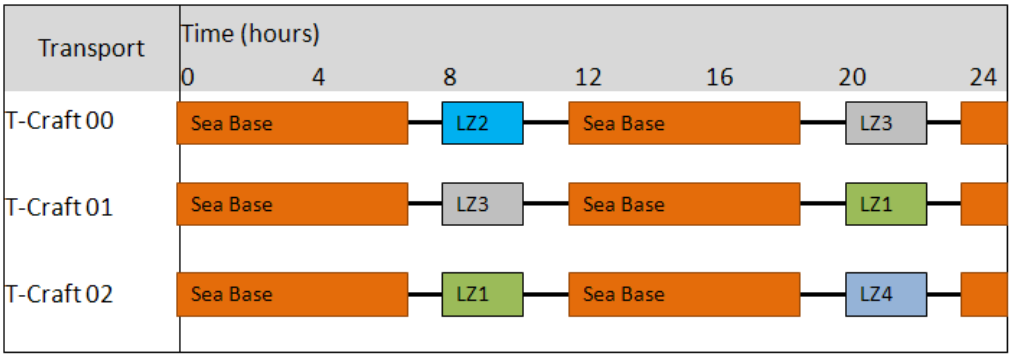


Figure 47: Scheduling Contrast Between Models

model takes a mandatory break after the vessel operates a number of hours. This is treating the operational limit as a repair requirement where once an operational time is exceeded, a maintenance period is required. This assumption difference impacts the consideration of the days as independent or grouped. In the case of the two days considered together, the rest period will continue into the second day by the amount of time the operational limit was exceeded. If the two days are considered separately, the vessel operations start immediately at the start of day 2.

11.3 Results Comparison

The CDM study examined the total amount of cargo delivered, based on a percent of the total demand as well as the tracking of the individual assets given in the schedules. CDM treats the days separately so the data presented here will be individual days and the results are given in Table 16. The days could be handled sequentially within the model and these results were analyzed, but not included here to keep the assumptions as close as possible.

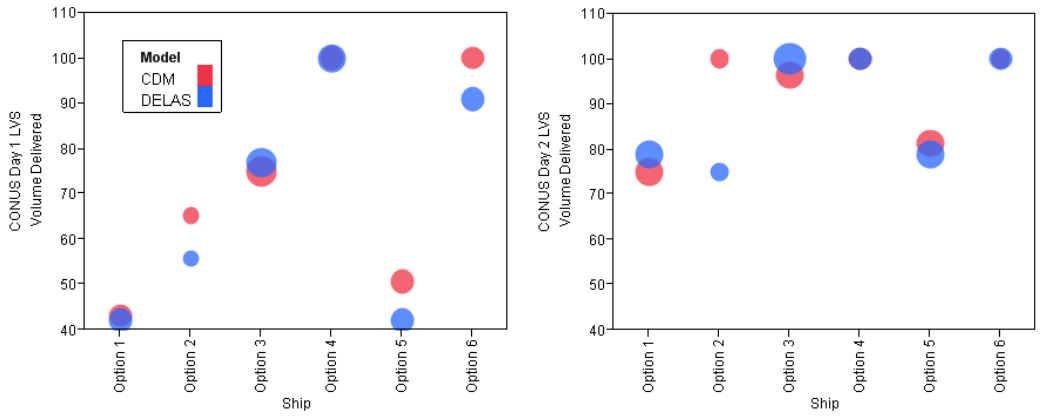
The results are plotted in Figure 48 through Figure 51. The percent of cargo delivered is compared with the size of the bubbles representing the number of vessels unloaded at the shore.

Table 15: CDM Results

Location	Cargo Type	Ship	Day 1 Unloaded	Day 2 Unloaded	Day 1 % Vol	Day 2 % Vol
Domestic	LVS	Option 1	5	6	43.2	75
		Option 2	3	3	65.2	100
		Option 3	9	6	75	96.4
		Option 4	6	4	100	100
		Option 5	6	6	50.8	81.3
		Option 6	5	3	100	100
		SSC	27	27	45.04	100
	Tractor-Trailer	Option 1	6	6	62.4	97.6
		Option 2	3	3	55.1	87.2
		Option 3	8	6	100	100
		Option 4	6	4	100	100
		Option 5	9	6	75	97.6
		Option 6	4	4	100	100
		SSC	27	27	53.46	84.94
International	LVS	Option 1	6	3	74.5	60.8
		Option 2	3	2	100	100
		Option 3	5	5	100	100
		Option 4	3	3	100	100
		Option 5	6	5	100	100
		Option 6				
		SSC	16	14	90.22	100
	Tractor-Trailer	Option 1	5	5	100	100
		Option 2	3	3	100	100
		Option 3	5	4	100	100
		Option 4	3	3	100	100
		Option 5	5	5	100	100
		Option 6				
		SSC	16	14	100	100

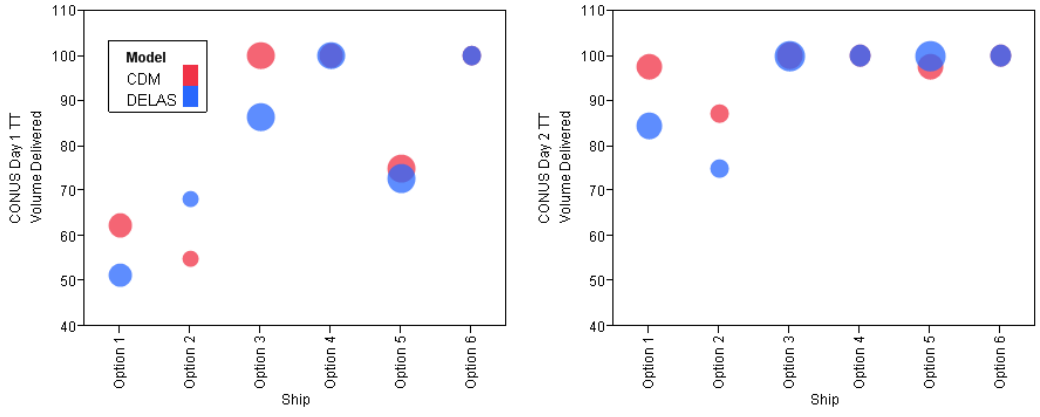
Table 16: DELAS Results

Location	Cargo Type	Ship	Unloaded Day 1	Unloaded Day 2	Day 1 % Vol	Day 2 % Vol
Dom	LVS	Option 1	6	6	42.3	78.9
		Option 2	3	3	55.8	75.0
		Option 3	9	8	76.9	100.0
		Option 4	8	4	100.0	100.0
		Option 5	6	6	42.3	78.9
		Option 6	6	4	91.0	100.0
		SSC	46	36	61.5	100.0
	Tractor-Trailer	Option 1	6	6	51.5	84.4
		Option 2	3	3	68.2	75.0
		Option 3	9	8	86.4	100.0
		Option 4	8	4	100.0	100.0
		Option 5	9	8	72.7	100.0
		Option 6	4	4	100.0	100.0
		SSC	46	28	78.8	100.0
Int	LVS	Option 1	6	5	75.0	100.0
		Option 2	3	3	60.0	60.0
		Option 3	9	5	97.5	100.0
		Option 4	5	5	100.0	100.0
		Option 5	6	5	75.0	100.0
		Option 6	5	5	100.0	100.0
		SSC	35	15	100.0	100.0
	Tractor-Trailer	Option 1	6	5	88.6	100.0
		Option 2	3	3	60.0	60.0
		Option 3	5	5	100.0	100.0
		Option 4	5	5	100.0	100.0
		Option 5	9	5	97.1	100.0
		Option 6	5	5	100.0	100.0
		SSC	30	15	100.0	100.0



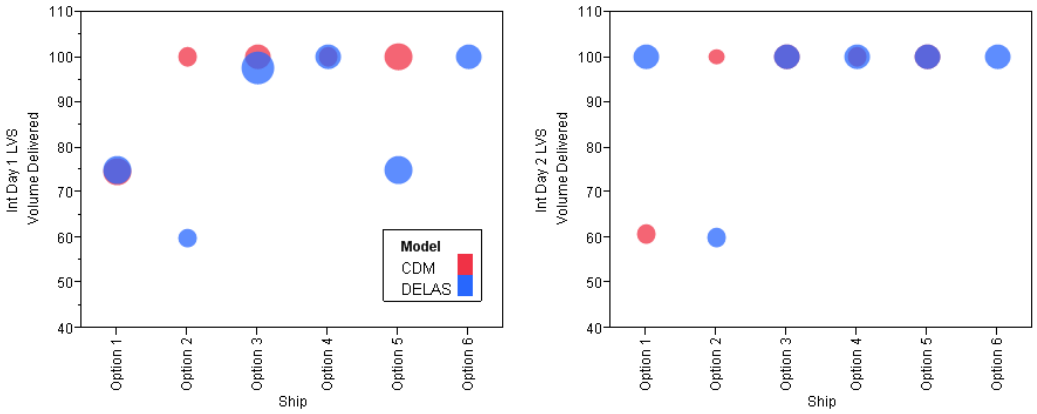
(a) Day 1 (b) Day 2

Figure 48: CONUS with LVS



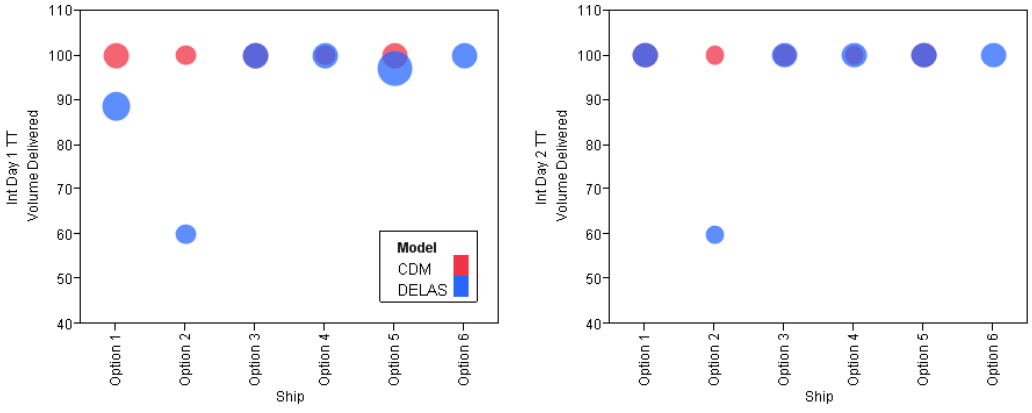
(a) Day 1 (b) Day 2

Figure 49: CONUS with Tractor-Trailer



(a) Day 1 (b) Day 2

Figure 50: International with LVS



(a) Day 1 (b) Day 2

Figure 51: International with Tractor Trailer

11.4 Discussion of Results

Overall, the results generated by this model and by CDM were comparable. There were a few discrepancies, which will be discussed, but overall, it is decided that this model demonstrated the same results and trends of the commercially developed CDM tool. In general, the CDM results have a slightly higher percentage, which can be attributed to CDMs use of partial containers. The total demand for this model is greater because it was rounded up to full containers. The percent delivered is calculated as the percentage of each type of container.

One interesting results is the scheduling conflict presented in Figure 46 was resolved in that Option 1 for CONUS with LVS has six completed trips instead of five given in the schedule. Figure 48 shows this with the size of the bubble for Option 1 being larger, but the overall performance is not significantly different. Option 1 has the most variance in results because the sensitivity to the schedule.

The largest discrepancy is with Option 1 and 2, with the trends flipping between the data generated with this works model and CDMs data. These differences are due to the difference in scheduling, seen in Figure 52 which impacts Figure 50(b). The other discrepancies with these two option are due to cargo selection, with the same number of connector being unloaded, but the amount of cargo differing significantly. These two options have more complex geometries than the other four. It is impossible to further explain the loading discrepancies without knowledge of the algorithm used by CDM to load the ships.

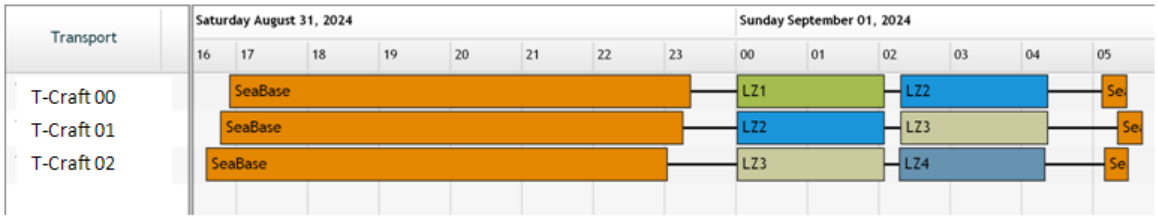


Figure 52: Additional Scheduling Shortcoming

Where schedule formulation does not drive the number of trips completed, the schedules can be matched. Figure 53 and 54 show the movement matches for the two models when option 3 is studied in more depth. The slight variation in the cargo delivered is due to the selection of beach to visit. This choice is based on the cargo selection algorithm and the distance to travel. The exact distances were not provided, only a range of distances. The variation in beach selected also impacts the cargo delivered since the first two trips to all beaches are now included and only one third trip as opposed to three of each trip for the CDM schedule.

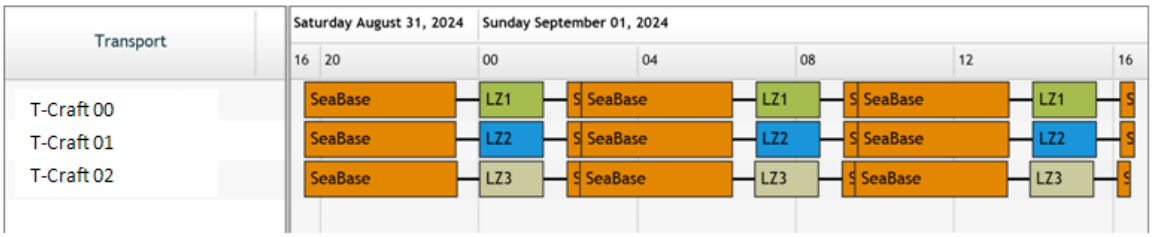


Figure 53: CDM Scheduling Option 2

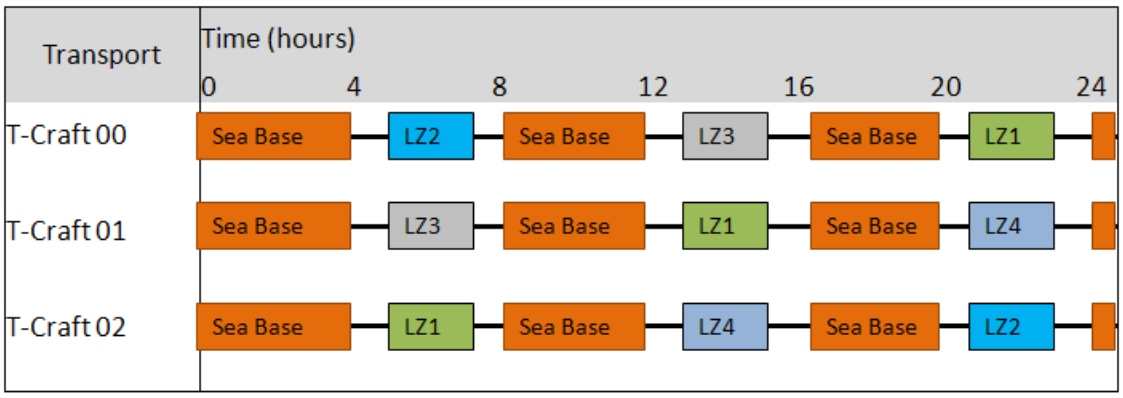


Figure 54: DELAS Scheduling Option 2

The discrepancies between CDM's results and the model developed for this thesis have been discussed and rationalized. This chapter documented the assumptions made by CDM for this study and how those differed from this models implementation. Details of the cargo selection algorithm for CDM were not available, but this results

in some variation. The scheduling algorithm is external for CDM and internal for DELAS, but the movements matched except in cases when the CDM algorithm introduced additional queuing. This comparison validates the connector process within the DELAS model.

CHAPTER XII

CAPABILITIES SUMMARY

The purpose of this chapter is to highlight the capabilities of a model incorporating the techniques discussed thus far in this thesis and to compare those to the gaps described in section 2.4. From here forward, the model will be known as DELAS (Discrete Event Logistics Advanced Simulation), representing the final model incorporating the formulations described in Chapters 6 through 10.

12.1 Addressing Gaps in Previous Models

Chapter 2 detailed existing logistics models which were limited applications of the needed components of the Sea Base model, but gaps in the application and methods remained which will serve as the basis for the work presented in this thesis. Previous work has demonstrated one or a few of these concepts but fail to generalize the models to a level where all these concepts can be incorporated into a single analysis. These gaps are:

- Breakdown of modeling problem into component
- Parametric scenarios
- Heterogeneous, interacting fleet
- Dynamic loading
- Dynamic routing
- Analyzing design requirements across multiple scenarios

The DELAS model addresses the first five gaps and will be applied in the remainder of the thesis to demonstrate the last gap. The model is broken down into the scenario, loading, and routing sub-problems. The scenario is inputted as a set of inputs and vectors. The use of vectors and matrices to handle information enabled the scenarios and the incorporation of the heterogeneous fleet. The fleet properties are monitored through matrices instead of describing the individual vessels behaviors within the discrete event process. The loading and routing subproblems are callable subroutines, enabled by the use of SimPy. The dynamic loading subroutine features an assignment problem added to a knapsack algorithm to intelligently select the cargo to load. The dynamic routing algorithm predicts the queues at the load and unload locations to determine the best load and unload locations to use on the next trip. The cargo to load is considered in this algorithm but reconsidered, as well as reconsidering the interface to use to load cargo, upon arrival at the load location. The case of multiple unload location, or hopping between beaches, is a special routing case. A traveling salesman problem is added to the cargo selection algorithm to determine if it is advantageous to unload at multiple locations. This algorithm replaces the cargo selection within the routing algorithm. The complete DELAS model is able to capture a variety of operations with a heterogeneous fleet. The complete code for the DELAS model is given in Appendix B.

12.2 Increased Capabilities

Addressing the existing gaps increases the capabilities that can now be modeled and many trade-offs become possible. The traditional design parameters of any vessel can now be evaluated at a system-of-systems level. These trade-offs now include the design choices of interface capabilities, which could not be addressed by existing models. Fleet level parameters can now be varied as well, including the fleet mix. The geographical parameters are now inputs, including the number of landing spots and

possibility of an intermediary port. The distances between points, such as staging bases are now inputs, as well as the properties of these bases. The locations of cargo objects can be specified, such as having multiple Sea Base groups. The impact of these parameters can now be analyzed for a variety of scenarios.

In this thesis, the focus is on overall system-of-systems metrics, but intermediary results can be extracted and analyzed. Such results include and utilization of individual or types of vessels, times in queues, and cargo load out weights and area. The next example will show intermediary results including routes and cargo selection. Having extractable information at various levels provides greater clarity into the actions within the simulation as well as supporting decision making at various levels, from vessel design to operation planning.

The following sections summarize the capabilities gained by incorporating each of the techniques described in Chapters 6 through 9. As the capabilities increase, an example scenario is developed to demonstrate these additions. The impact on the results as well as what type of results can be abstracted are detailed.

12.2.1 Matrix Formulation

The incorporation of matrices as a data handling formation allows for cargo objects to be treated as individual objects. In addition, the matrix formulation creates a more scalable model formulation and allows for the abstraction of sub-problems. By changing the formulation, the cargo aboard the individual assets can be tracked. The cargo transfer interface properties are now design variables to study the impact of incorporating additional interfaces. Figure 13 showed the impact of shifting from hard-coded if-then statements to matrix decisions. Note that these formulations can only be compared for the number of connectors unloaded at the shore. The matrix formulation results allow the cargo unloaded, it terms of weight, volume or type to be tracked. Loadings must be specified for each type of connector.

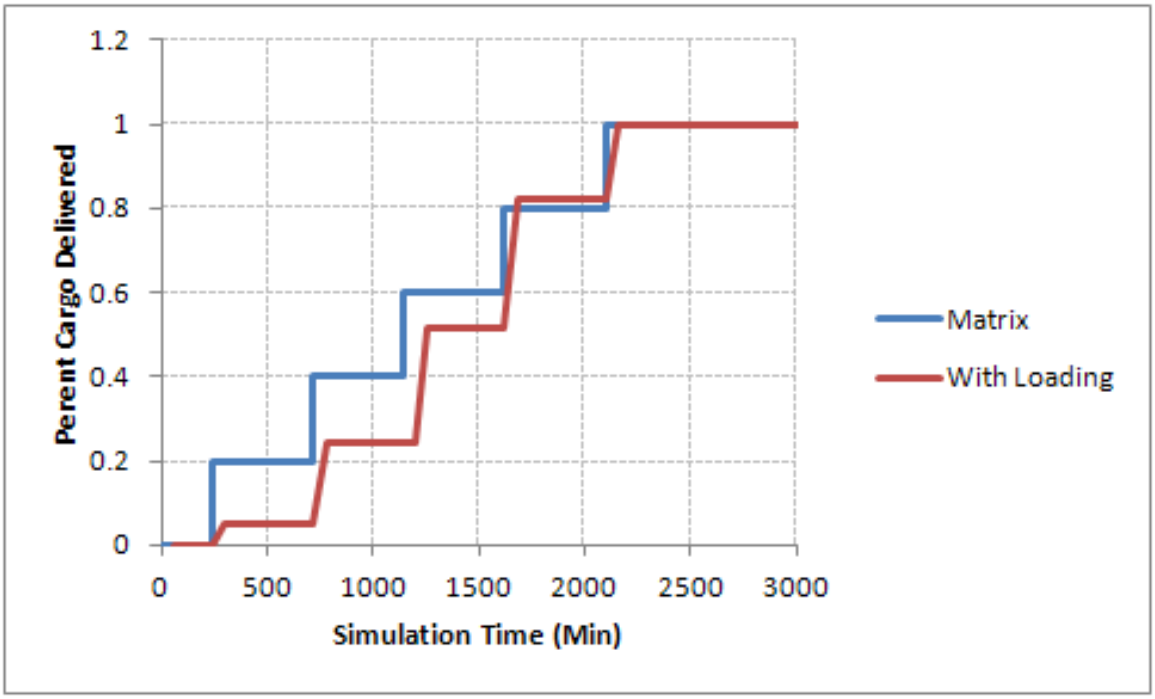


Figure 55: Matrix without and With Loading

12.2.2 Dynamic Loading

Dynamic loading allows for the investigation of different cargo demands. This enables the investigation of scenarios by varying the cargo demand, in addition to the geographical properties, i.e. distances and beach landing spots. Dynamic loading gives a more realistic delivery profile. Figure 55 compares the results for the matrix based formulation and incorporating dynamic loading. For the matrix formulation, it was assumed that the cargo was evenly divided between the 15 trips required for delivery. The dynamic loading loads the Marine Expeditionary Brigade (MEB) based on the weight and area constraints. Although the unloads occur at the same time, seen by the corresponding jumps, the loading makes the raise in percent weight delivered vary between the loads. The greatest advantage in incorporating the loading is the internal determination of the loading, without having to provide information beyond the cargo demand and physical characteristics of the connector.

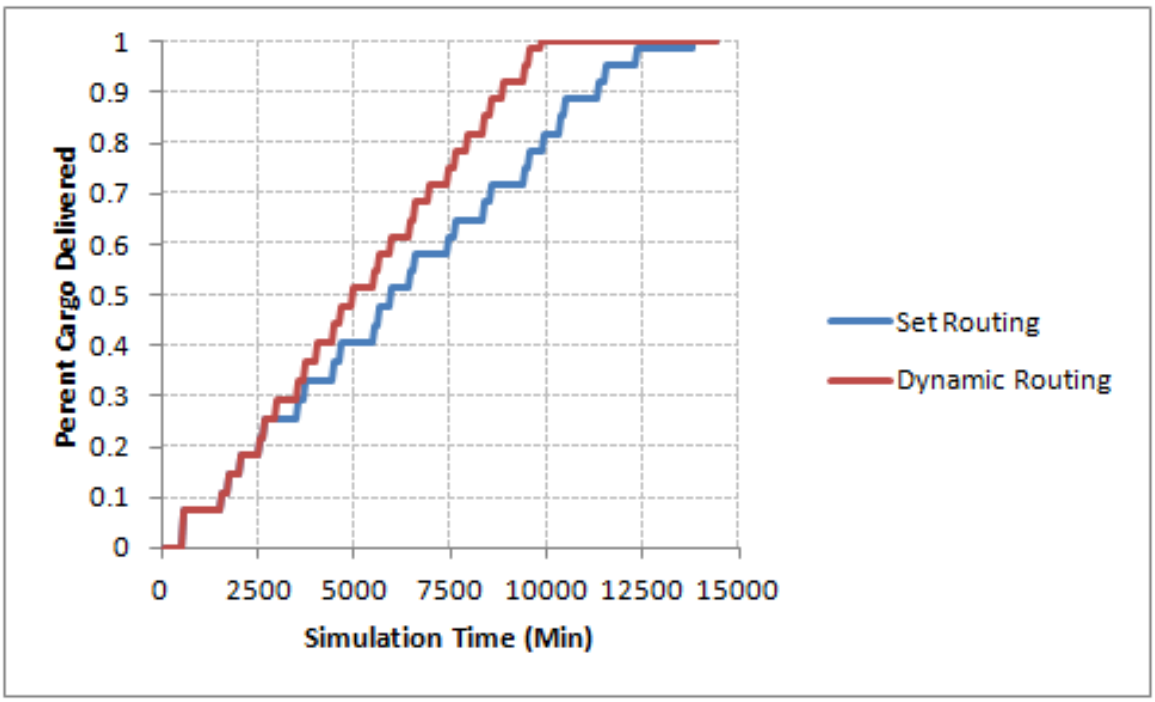


Figure 56: Set Routing vs Predictive Routing

12.2.3 Dynamic Routing

Dynamic routing allows the supply chain to be a function of the assets and cargo supply nodes. Figure 56 shows the improvement in the performance with the incorporation of dynamic routing. This comparison is for the delivery of a MEB from two LMSRs and one ISB, where the set routing sends one of the three MECs back and forth to the ISB. In addition to a performance improvement, dynamic routing enable a number of trade-offs. The number of cargo ships can be traded with the location and number of ISBs. Operational decisions can be investigated, including the use of in intermediary port as a cargo transfer node.

12.2.3.1 Multiple Unload Points

The option to unload at multiple cargo points, or hop between beached is an operational decision. Including this option introduces the capability for investigation of operational decision and rules. The scenario used is detailed further in Section 12.4

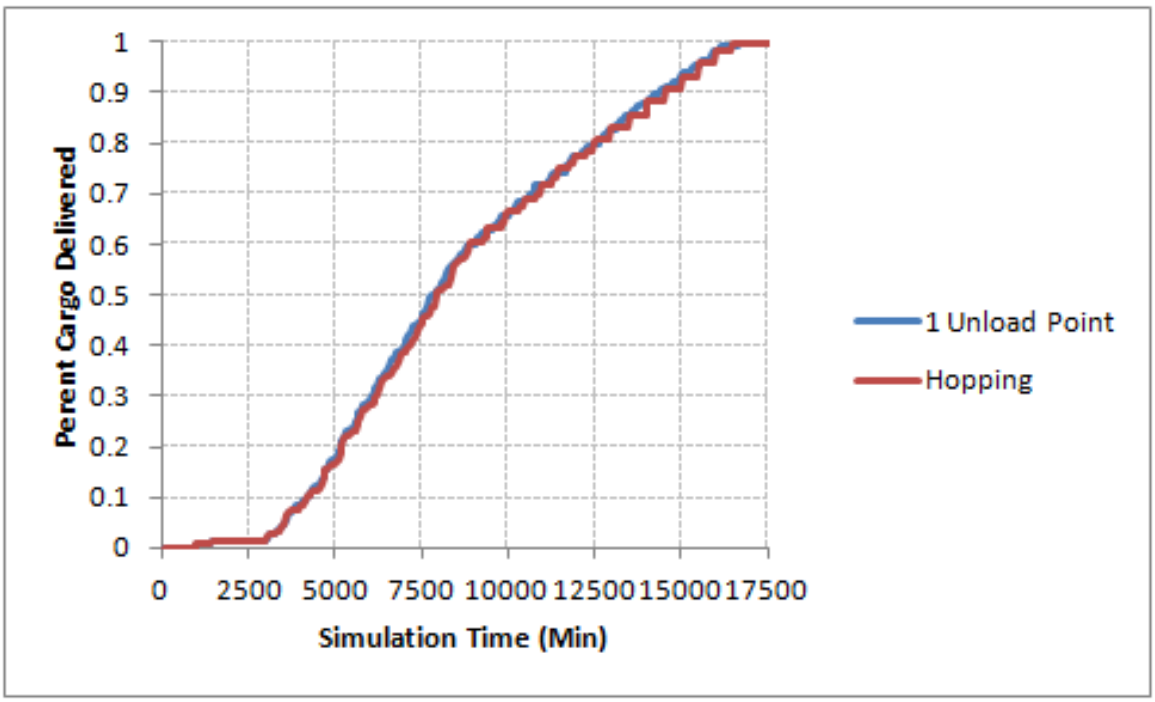


Figure 57: One vs Multiple Unload Points

as it is used in the full scale application example. Figure 57 compares the results with and without hopping. While the hopping results lag the single unload point through the middle portion of the simulation, it completes the delivery two hours earlier. The gain of hopping is seen when the cargo demand for each beach is less than the carrying capability of a single MEC.

12.3 Scalability

Section 7.3 described the scalability of the matrix formulation, which is only limited by run time. The general matrix formulation does not limit the size of the operation to model with the possible exception of if the matrix became so large the minimum value could not be found. The more limiting element is the MILP for the dynamic loading and routing with multiple unload points. The dynamic routing algorithm run time increases with the number of cargo supply objects and cargo types. As described in Section 9.7, the number of variables in the multiple unload point algorithm grows

Table 17: Vessels and Vehicles

Vessel/ Vehicle	Total Number
MEC	3
MLP	2
LCAC	4
LMSR	3
LHD	3
CH46	4
CH53	6

O(n(n+1)), limiting the number of beach groups. If the run time for this MILP becomes unacceptable using the brand and bound algorithm in LPSolve55 or the exact solution is no longer identifiable, heuristics can be implemented to find acceptable solutions in a reasonable run time. The major obstacle to expanding the operation modeled is the increase in run time.

12.4 Full Scale Application Example

Thus far, the examples executed by the model have been small scale with a subset of the models capabilities to test and demonstrate algorithms within the model. This section will present a basic example on the full capability of the DELAS model. Section 6.2.1.1 describes a large scale military operation. The vessels and vehicles involved are given in Table 17 and initially distributed between two ISBs. Initially the infantry is delivered to shore to secure two beachheads with landing zones given in Table 18. The MEC is the only vessel that has the option to unload at multiple beachheads. The next day begins the delivery of tanks and heavy artillery. Day 3 brings the resupply of food and water on pallets. This leads to the cargo demand schedule, for each beach group, in Table 19. The distances between the locations is given in Table 20. The LHDs are located closer to the shore than the rest of the Sea Base to deploy the helicopters, called the SB close distance in the input table.

DELAS was run for 20 days, requiring 2.5 minutes, and results in the time history

Table 18: Beach Landing Zones

Beach Type	Group 1	Group 2
MEC	3	3
LCAC	2	2
Helo	2	2
Austere Port	1	1

Table 19: Demand Schedule per Beach Group

Cargo Category	Initial	Day 1	Day 2
Marine Platoon	78		
EFV		47	
LAV-25		27	
M1A2		47	
Arty Element		21	
Mortar Element		12	
Antiarmor Element		12	
Pallet			3000

Table 20: Vessels and Vehicles

Distances (nmi)	SB	SB close	BeachGroup1	BeachGroup2	ISB00	ISB01
SB	0	25	100	120	500	750
SB close	25	0	75	80	500	750
BeachGroup1	100	75	0	20	600	850
BeachGroup2	120	80	20	0	620	870
ISB00	500	500	600	620	0	0
ISB01	750	750	850	870	0	0

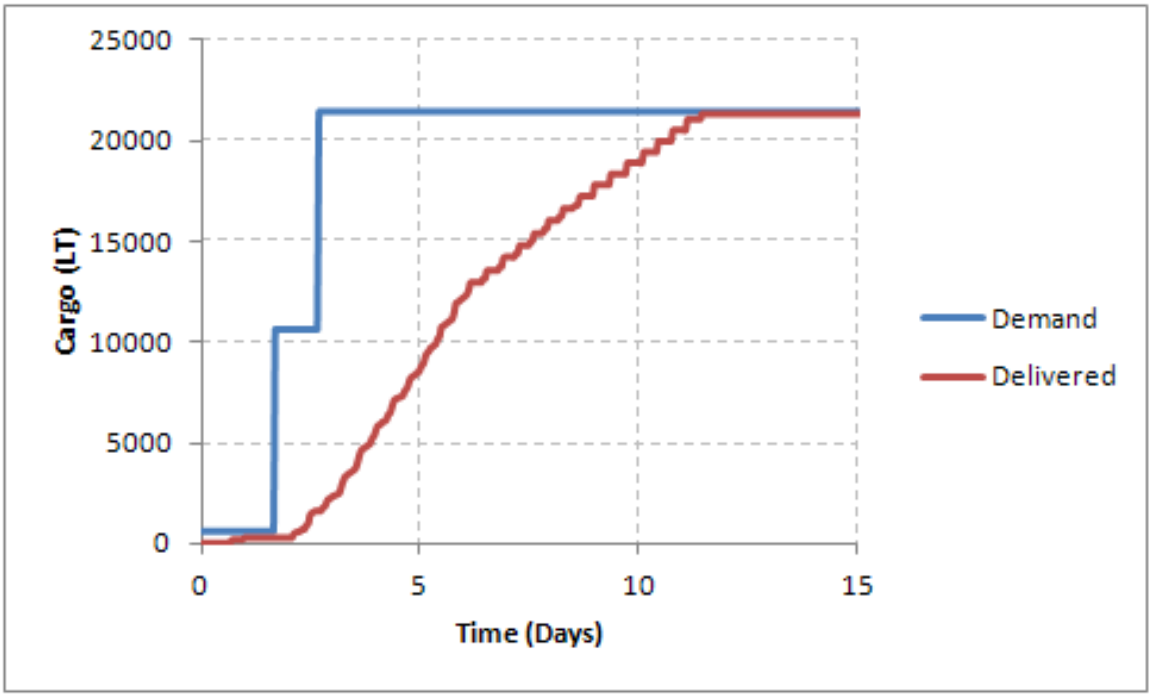


Figure 58: Time History Results

in Figure 58, showing the demand versus cargo delivered over time. The knee in the curve between Day 6 and 7 results from the cargo requirements. At this point, the helicopters have brought to shore everything they can fit and lift, with the remaining cargo requiring movement by surface connectors. The discrete arrivals of surface connectors is visible in the flatter portion of the curve. Figure 59 shows the time history for shortfall, which accumulated every 8 hours for this example. This time period translates to a penalty if the demanded cargo is not delivered within 8 hours of the demand signal.

To give an idea of the movement within the model, a selection of the routing decisions, including predicted cargo to carry, made within the model are presented in Table 22. In total, there are 606 predicted missions. A sample of the actual routing of the connectors is given in Table 23. There are 1197 total routes, when the connector travels from one location to another. The delivery of the troop by MEC is seen initially as they are the only connectors that deploy from the ISBs. Once the

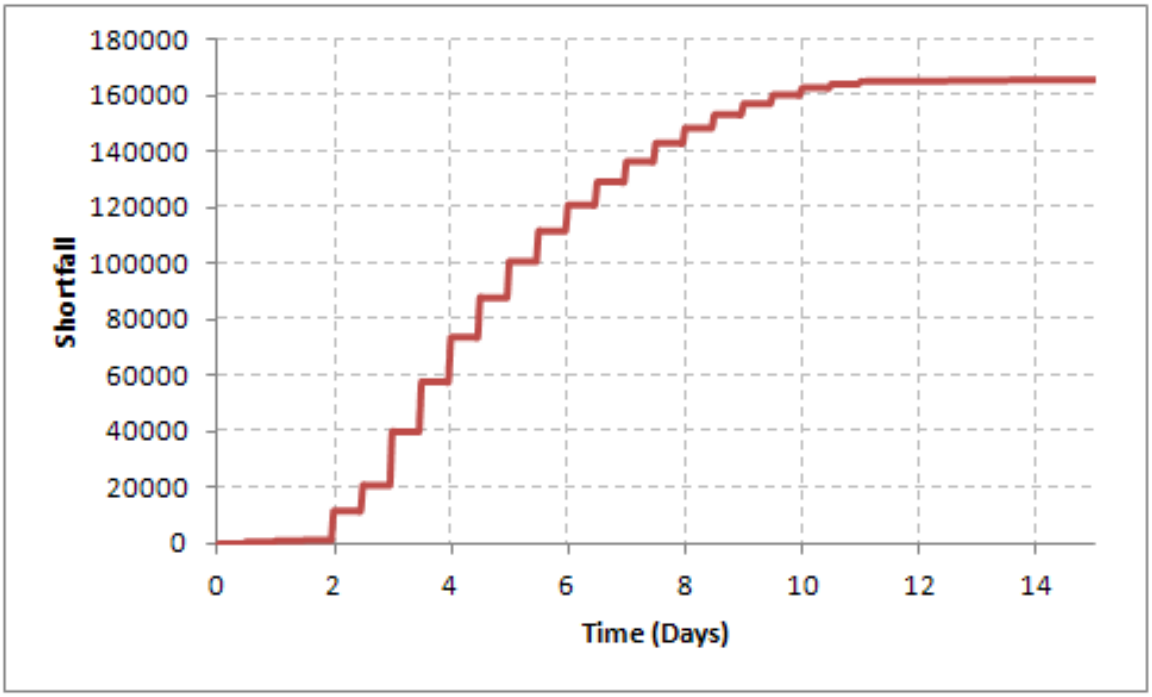


Figure 59: Shortfall Results

Table 21: Total Trips per Type of Asset

Connector	Number of Trips
MEC	85
LCAC	14
CH46	180
CH53	313

LMSRs, MLPs, and LHD have arrived at the Sea Base, the MEC are joined by the LCACs and helicopters to deliver cargo. The total number of trips by each type of asset are given in Table 21. The total number of unloaded is smaller than the number of predicted missions, indicating not all missions could or needed to be completed.

12.5 Performance Comparison and Trade-offs

To capture some of the trade-offs capable with this model, a basic design trade-off is explored in this section. For the four scenarios described in Chapter 6, five candidate designs were selected and DELAS model run. These runs were repeated for one and ten MECs available. The properties of the designs selected are given in Table 24.

Table 22: Predicted Routing Results

Vessel	Time (min)	Start	Load	Unload	Platoon	EFV	LAV - 25	M1A2	Arty	Mortar	Antiarmor	Pallet
MEC00	1	start00	ISB00	Beach1	23	0	0	0	0	0	0	0
MEC01	1	start01	ISB01	Beach1	23	0	0	0	0	0	0	0
MEC02	1	start00	ISB00	Beach2	23	0	0	0	0	0	0	0
MEC00	1019.5	Beach1	ISB00	Beach2	23	0	0	0	0	0	0	0
MEC02	1019.5	Beach1	ISB00	Beach2	23	0	0	0	0	0	0	0
MEC01	1394.5	Beach1	ISB00	Beach2	23	0	0	0	0	0	0	0
CH4600	3005	SBstart	SB close	Beach1	0	0	0	0	0	1	0	0
CH4601	3005	SBstart	SB close	Beach1	0	0	0	0	0	1	0	0
CH4602	3005	SBstart	SB close	Beach2	0	0	0	0	0	1	0	0
CH4603	3005	SBstart	SB close	Beach2	0	0	0	0	0	1	0	0
CH5300	3005	SBstart	SB close	Beach1	0	0	1	0	0	0	0	0
CH5303	3005	SBstart	SB close	Beach1	0	0	1	0	0	0	0	0
CH5305	3005	SBstart	SB close	Beach2	0	0	1	0	0	0	0	0
CH5302	3005	SBstart	SB close	Beach2	0	0	1	0	0	0	0	0
CH5304	3005	SBstart	SB close	Beach1	0	0	1	0	0	0	0	0
CH5301	3005	SBstart	SB close	Beach1	0	0	1	0	0	0	0	0
MEC00	3056.5	Beach2	SB	Beach1, Beach2	3	0	1	0	19	16	1	0
MEC02	3056.5	Beach2	SB	Beach1, Beach2	6	0	0	0	20	1	6	0

Table 23: Actual Routing Results

Vessel	Time (min)	Start	End	Platoon	EFV	LAV - 25	M1A2	Arty	Mortar	Antiarmor	Pallet
MEC00	1	start00	ISB00	0	0	0	0	0	0	0	0
MEC01	1	start01	ISB01	0	0	0	0	0	0	0	0
MEC02	1	start00	ISB00	0	0	0	0	0	0	0	0
MEC00	989.5	ISB00	Beach1	23	0	0	0	0	0	0	0
MEC02	989.5	ISB00	Beach1	23	0	0	0	0	0	0	0
MEC01	1364.5	ISB01	Beach1	23	0	0	0	0	0	0	0
MEC00	2008	Beach1	ISB00	0	0	0	0	0	0	0	0
MEC02	2008	Beach1	ISB00	0	0	0	0	0	0	0	0
MEC01	2383	Beach1	ISB00	0	0	0	0	0	0	0	0
CH4600	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH4601	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH4602	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH4603	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH5300	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH5303	3005	SBstart	SB close	0	0	0	0	0	0	0	0
CH5305	3025	SBstart	SB close	0	0	0	0	0	0	0	0
CH5301	3025	SBstart	SB close	0	0	0	0	0	0	0	0
CH5302	3025	SBstart	SB close	0	0	0	0	0	0	0	0
CH5304	3025	SBstart	SB close	0	0	0	0	0	0	0	0
MEC00	3026.5	ISB00	Beach2	23	0	0	0	0	0	0	0
MEC02	3026.5	ISB00	Beach2	23	0	0	0	0	0	0	0
CH5300	3055	SB close	Beach1	0	0	1	0	0	0	0	0
CH5303	3055	SB close	Beach1	0	0	1	0	0	0	0	0
CH4602	3061.923	SB close	Beach2	0	0	0	0	0	1	0	0
CH4603	3061.923	SB close	Beach2	0	0	0	0	0	1	0	0
CH4601	3065	SB close	Beach1	0	0	0	0	0	1	0	0
CH4600	3065	SB close	Beach1	0	0	0	0	0	1	0	0
CH5304	3075	SB close	Beach1	0	0	1	0	0	0	0	0
CH5301	3075	SB close	Beach1	0	0	1	0	0	0	0	0
CH5305	3077	SB close	Beach2	0	0	1	0	0	0	0	0
CH5302	3077	SB close	Beach2	0	0	1	0	0	0	0	0

Table 24: MEC Candidate Designs

Vessel Name	Large	Large, Fast	Medium	Small	Small, Slow
SES speed (kts)	25	45	35	45	25
ACV speed (kts)	2	2	5	10	10
Transition time (min)	45	45	30	15	15
Transition distance (nmi)	2	2	2	2	2
Max lift (LT)	800	800	550	300	300
Max area (sqft)	14000	14000	8000	2000	2000
Range (nmi)	6000	6000	6000	6000	6000
Mean time between failure (min)	1440	1440	720	360	360
Mean time to repair (min)	300	300	200	100	100
Use Port Ramp?	1	1	1	0	0
Time to load (min)	450	450	450	450	450
Use Stern Ramp?	1	1	1	1	1
Time to load (min)	300	300	300	300	300
Use Crane?	1	1	0	0	0
Time to load (min)	600	600	600	600	600
Use Austere Port?	1	1	0	0	0
Time to unload (min)	120	120	0	0	0
Beach time to unload (min)	180	180	180	180	180

These five designs approximate realistic design trades, while investigating the impact of speed and size. The large connector has a greater transition and loading times, but has more interface options.

Figure 60 and 61 show radar plots of the performance of the five connectors across the four scenarios. The designs that maximize the area perform better across all of the scenarios. The smaller the shortfall, the better the performance, with zero shortfall on the outer edge of the radar plot. The smaller difference in the shortfall values for the small military and humanitarian scenarios indicate the MEC design does not have a large effect on the operational performance. The design of the MEC has a greater impact on the performance in the major combat and sustainment operations. Figure 60 shows the sustainment operation is better served by a slower ship, with a slight improvement with a larger ship. This effect is caused by the MEC out cruising the rest of the fleet. Since recurring demand does not occur until 24 hours after the first

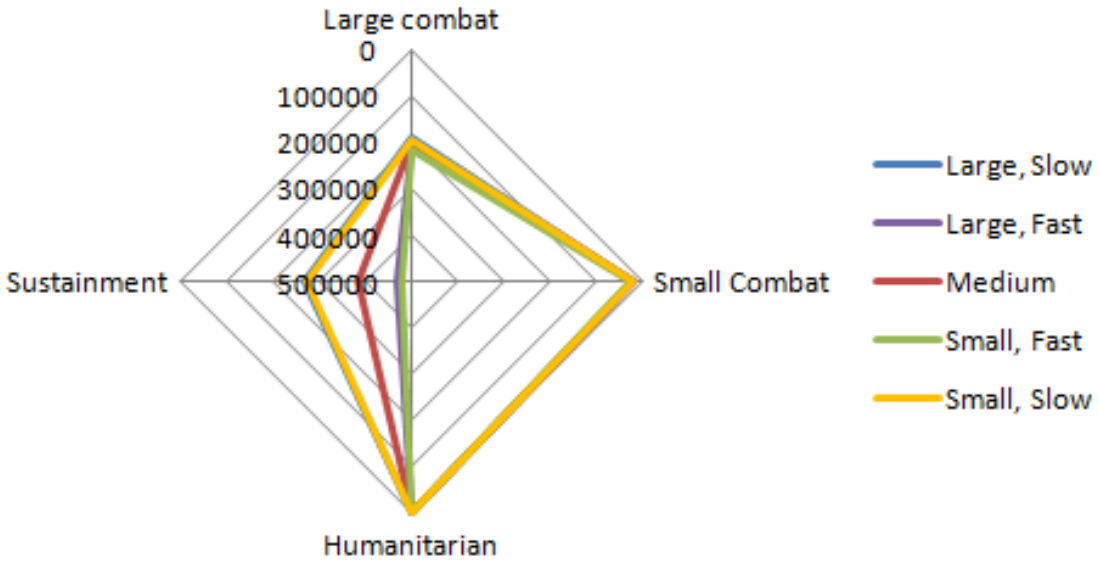


Figure 60: Radar Plot for 1 MEC

delivery, the demand occurs faster if the MEC is faster. The operational assumption was made to simulate the phases of the operation, for example, the resupply demand for a MCO would not occur if the first wave of troops were not on shore yet. The fast MEC leads the cargo supply ships and the second delivery is more challenging. The closeness of the results for MCO indicates a single MEC does not have a large impact on the overall performance. Figure 61 shows the impact of including a large number of MECs into the scenarios. Very little performance is gained with additional MECs for the small combat and humanitarian scenarios. The impact of the larger MEC is exaggerated when more are included, seen in the MCO results. Based on these assumptions, ten large, slow MECs should be incorporated into the future fleet.

To visualize the impact of changing just one assumption, the assumption that demand does not start until the first arrival was modified. The first recurring demand now occurs 24 hours after the start of simulation time, instead of 24 hours after first connector arrival. The results with one MEC collapse to a much larger shortfall in the sustainment operation due to the much higher cargo demand. Figure 62 shows the results with 10 MECs. The results are closer to what is intuitive with the large,

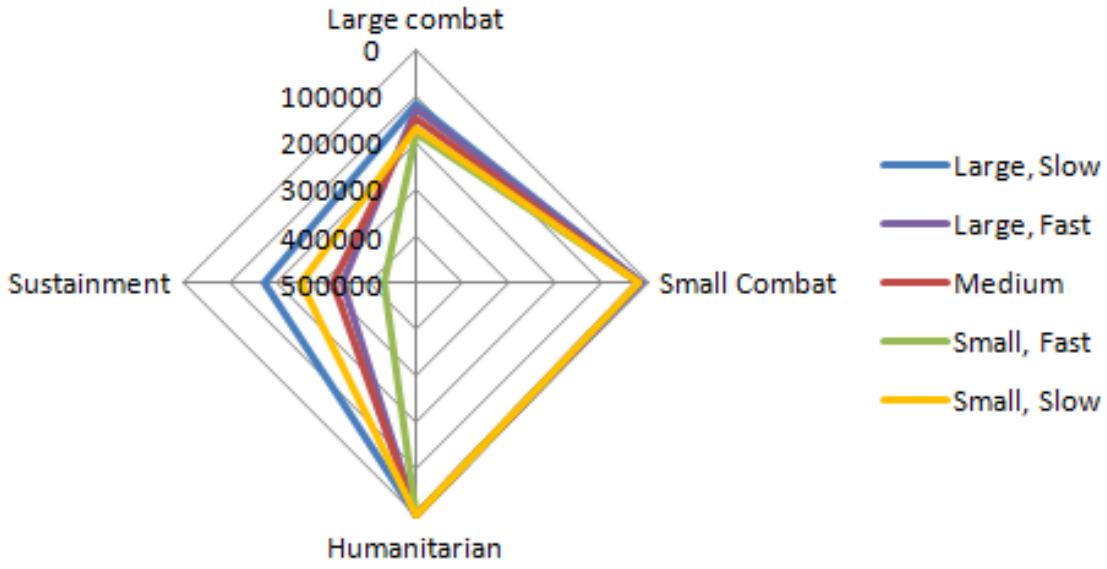


Figure 61: Radar Plot for 10 MECs

fast MEC providing the greatest benefit.

One set of trade-offs was explored here but the dependence on assumptions was also highlighted. Based on these four scenarios and the five candidate scenarios, initially ten large, slow MECs were recommended. This is based on the assumption of the vessels capabilities and the fleet mix present. The large vessel provides more flexibility in the interface options. Although taking longer to load and unload, this vessel delivers more cargo in a single trip. Modifying the demand generation assumption changes the recommendation to large, fast ships. These recommendations are based on a set of assumptions, including the usage of maximum speed during the long distance transits and not traveling with the fleet. The selection of underlaying assumptions is important in the exploration of design alternatives.

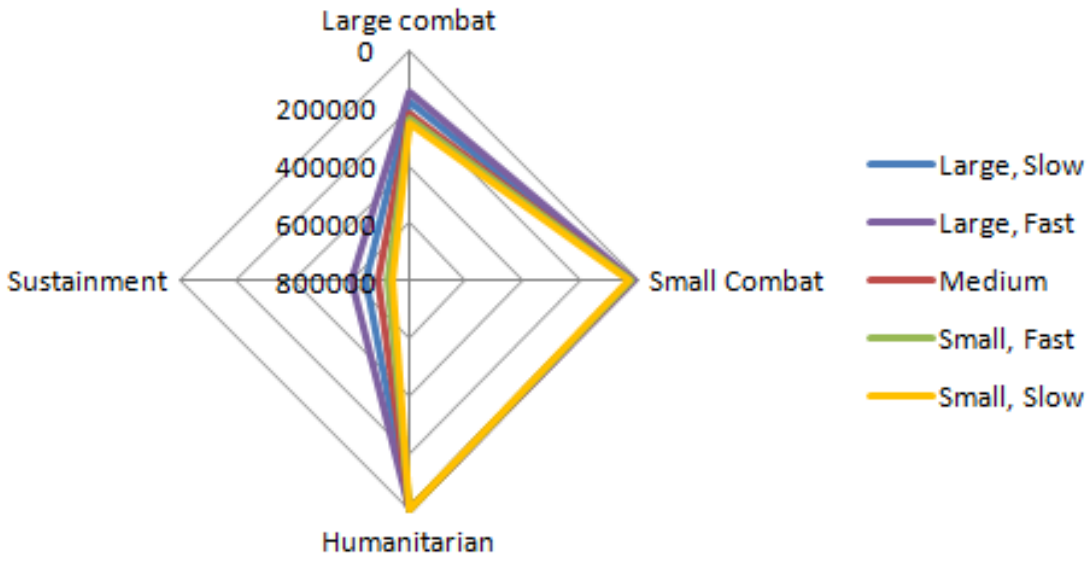


Figure 62: Radar Plot for 10 MECs - Scenario Definition Change

CHAPTER XIII

ROBUST DESIGN PROCESS

The previous chapters focused on the development of the DELAS model. The next section will apply that model for a robust ship design application. The next three chapters focus on the hypothesis that feasible scenario robust analysis can identify the design drivers for a range of scenarios. This chapter focuses on the background and process of robust design. Chapter 14 discusses the concept and formulation of feasible scenario analysis. Last, the results of the robust design analysis are discussed. Included in these chapter are the following experiments:

1. Identify measure of performance (MoPs) that are relevant to range of operations
2. Identify sampling methods for segmented spaces
3. Compare the feasibility of the sampling methods to complete coverage
4. Compare robustness results for complete coverage, and feasible scenario options
5. Identify the design drivers for the conceptual connector - MEC

13.1 Concept of Robust Design

The goal of robust design is to find the best values of the parameters that minimize the performance variability while keeping mean performance on a target [193]. There are three types of variability, or noise, that can be considered in robust design [146]:

1. External - environment of operation
2. Unit-to-unit - manufacturing variation

3. Deterioration - degradation of individual components

Since this thesis is focused on the design and analysis, the only noise that will be considered is external, due to the environment in which the conceptual vessel will operate. The noise factors will be identified in a later section.

Robust design can be traced back to Taguchi's work in the 1950s and 1960s, but has been criticized and expanded to address these concerns. The new focus is on understanding rather than solution driven analysis by varying control and factors in the same experiment to capture interactions [31]. Orthogonal arrays are the basis of Taguchi's method but are criticized for ignoring interaction effects, but design of experiments considering control and noise factors resolve this issue [60]. Combined arrays, such as modified central composite designs, have been shown to be more efficient [129]. Computer experiments have different demands than physical experiments leading to different experiment designs, such as latin hypercube [158]. Instead of minimizing a signal-to-noise ratio, the mean and variation are considered separately through the integration of response surface methodology [61]. This allows the mean and variation to be investigated seperately as well as combined into a loss function. It is difficult to optimize multi-responses in complex process since the equation pertaining to summing of weighted S/N ratio is difficult to explain from the vantage of Taguchis quality loss function [28].

13.1.1 Use of Robustness in Ship Design

Robust design was used in a ship design application by Scheibe [165], with decision factors including MEC design parameters and the number of MEC present. Within a humanitarian scenario modeled in ARENA, the noise variables included deck use, number of shore spots, probability of hit and sink, and attrition rate. The outputs used were time to complete, percent cargo delivered, and portion of craft destroyed. This lead to recommendations on the number of MEC and lift capability. Applying

robust design to this problem will require mixing level or variables not seen in the example work. A limited number of vessel design variables and scenario variables were included in Scheibe's work. Building on Scheibe's work, Cason [56] included many more vessel level variables.

13.2 Robust Design Process

The process of robust design for this work will use the process executed by Schiebe and documented by Sanchez [162, 160]. The framework for a robust design:

1. Select the performance measures
2. Specify a loss function
3. Identify the factors
4. Plan the experiment
5. Analyze the results
6. Select the design drivers

The next six sections will further develop the work necessary for each step in the robust design framework. This includes the information necessary for application for this thesis using the DELAS model.

13.2.1 Select the Performance Measures

The performance measures of interest that will be selected are representative of military operations of different types. The selected measure of performance (MoP) is shortfall, which is a function of the difference between cargo needed and cargo delivered over the time of the operation. The concept of shortfall is illustrated in Figure 10 where the total shortfall is the sum of the area between the shortfall and demand curves. The size of the time unit can be set to an interval of interest, for example,

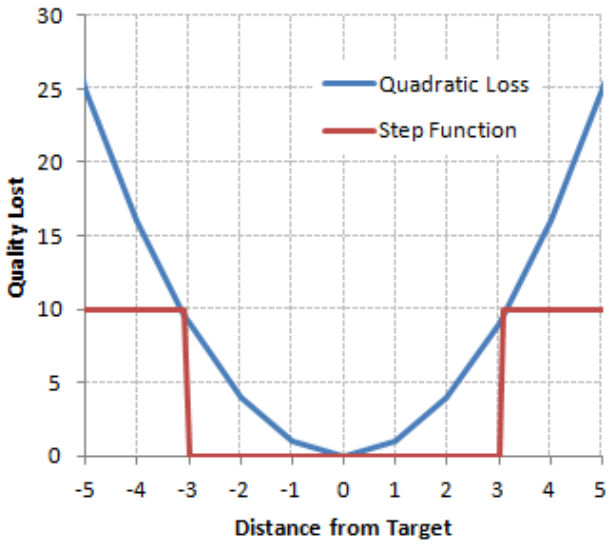


Figure 63: Loss Function

every day or every eight hours. In reality, cost must be considered, so the estimated fuel usage, a cost metric, is captured. The fuel usage is only that used by the sea and air connectors, not Sea Base supply ships and estimated based on inputted fuel usage in the modes of operation for each connector.

13.2.2 Specify Loss Function

Sanchez details a method for trading off performance mean and variability by examining the expected loss function, where c is a scaling constant and τ is a target state. [161]:

$$E(loss) = c[\sigma^2 + (\mu - \tau)^2]$$

A quadratic loss function was selected for this project because a tolerance window does not exist. If a specific tolerance was selected, a step function or combination of quadratic and step could be used. As seen in Figure 63, a step function does not penalize if within tolerance were the quadratic loss penalizes any deviation from target.

For shortcoming and fuel usage, the goal is to minimize the metrics. The loss

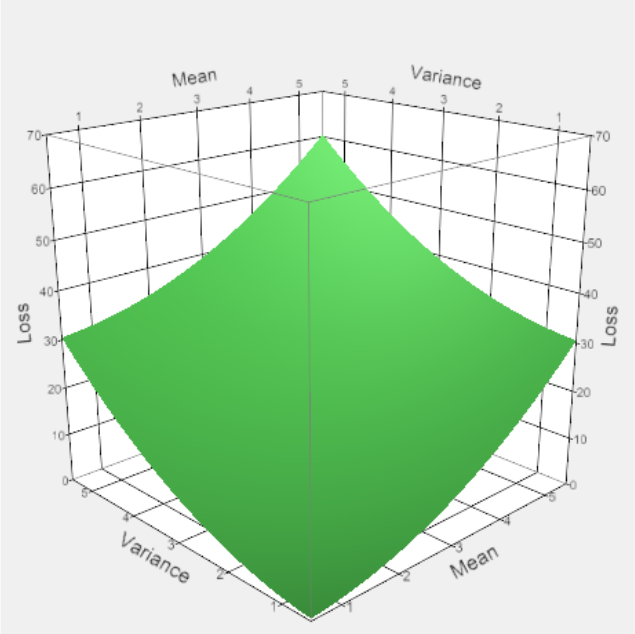


Figure 64: Loss Function Minimization

function then becomes:

$$E(loss) = c[\sigma^2 + \mu^2]$$

The quadratic loss is applied over the mean and deviation with the goal to minimize total loss, as seen in Figure 64. This leads to a trade-off in performance and variation in performance.

It is important to note that the units for shortcoming and fuel usage are not compatible for linear combination. The magnitudes are significantly different, so the loss function would be dominated by the fuel consumption. To correct the unit issue, the outputs will be normalized by the smallest value to non-dimensionalize the loss function components. This correction is only made when shortcoming and fuel consumption are combined into a single loss function. A weight can also be applied to each output based on the relative importance.

Table 25: Control Variables and Ranges

Vessel Variables	Min	Max
Number available	1	10
SES speed (kts)	25	50
ACV speed (kts)	2	10
Transition time (min)	10	45
Transition distance (nmi)	1	10
Max lift (LT)	250	800
Max area (sqft)	600	14000
Range (nmi)	2000	60000
Mean time between failure (min)	60	1440
Mean time to repair (min)	15	300
Use Port Ramp?	no	yes
Time to load (min)	60	600
Use Stern Ramp?	no	yes
Time to load (min)	60	600
Use Crane?	no	yes
Time to load (min)	120	1200
Use Austere Port?	no	yes
Time to unload (min)	20	300
Beach time to unload (min)	30	600

13.2.3 Identify the Factors

The factors selected for the robust design will focus on the MEC design and the operational scenarios. This will utilize a subset of the variables included in DELAS. The remaining parameters will be set to values representing a best guess of current and projected capabilities.

The control factors focus on the design and procurement options for the MEC. These include vessel design decisions and operational decisions. If the design decision would require the modification of an existing asset, it is assumed the change is possible, i.e. the use of the LMSRs side ramp to load cargo would require advances in the ramp system and would be included in the MEC design process if this interface is desired. The 18 control variables are listed in Table 25 with the ranges to be used in this study.

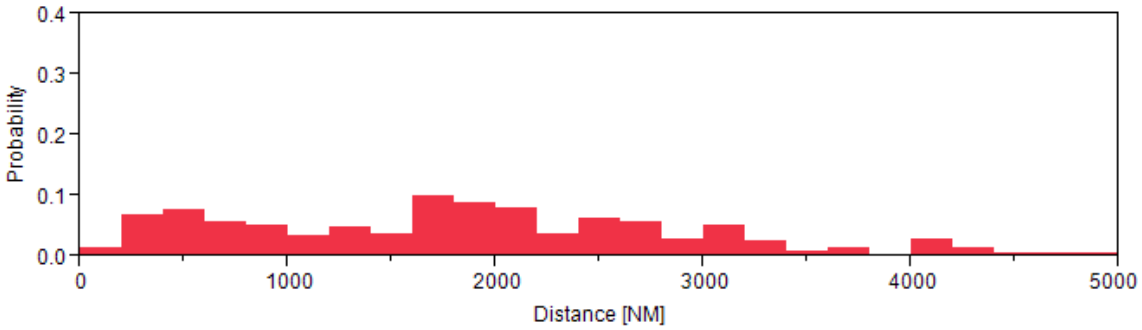


Figure 65: Distribution of Distances to ISB

The noise factors capture the variety of operational scenarios in which the MEC would participate. These factors are listed in Table 26. The distance to the ISB range was determined from work completed by Hyun Seop Lee in determining locations of potential conflicts. Five hundred random point were selected around the world and the distance from the nearest US base calculated based on usable seas, such as commerical shipping lanes. The distribution of distances is given in Figure 65 and form the range for this variable. The number of other vessesl present is an estimate of the maximum number of Navy assets that would be deployed to a single operation. The ranges for the cargo demand were derived from the scenarios discussed in Chapter 6 and represent the minimum and maximum values from the described scenarios. The pallet and petroleum pod demand will be a repeated demand, with the set number of pallets nad pods being demanded every 24 hours.

13.2.4 Plan the Experiment

As discussed in 4.5.1, the scenarios could be fully parametric with each variable that serves to define the scenario is treated as an independent continuous or discrete variable. The design of experiments would completely cover any possible scenario and serves as the basis for comparison. This full coverage option would lead to some unrealistic scenarios, such as bringing in only tanks and tents. This may lead to a biasing of the results based on unrealistic scenarios. The design space encompassing

Table 26: Noise Variables and Ranges

Variable	Min	Max
Distance to ISB (nmi)	200	5000
Sea Base distance (nmi)	25	200
Number of landing spots	1	5
Other vessels present:		
LMSR	0	3
LHD	0	11
LSD	0	11
LPD	0	11
MLP	0	6
LCACR	0	36
JHSV	0	4
MV22	0	5
CH46	0	11
CH53	0	11
Cargo damand:		
Marine Platoon	0	156
EFV	0	94
LAV - 25	0	54
HMMWV	0	12
M1A2	0	94
EFSS Element	0	6
CEB Element	0	48
Arty Element	0	42
Mortar Element	0	24
Antiarmor Element	0	24
HIMARS Element	0	6
Pallet	0	3000
TEU - Water	0	76
TEU - MRE	0	36
TEU - beans	0	5
TEU - shelter	0	25
TEU - Eng Co	0	5
Petroleum Pod (500 gal)	0	1000

the feasible scenarios can not be expected to be regular shaped. It is necessary to sample enough of the scenario space to encompass the uncertainty in future operations without sampling unrealistic space. The concept of feasible design space and the sampling of the design space compared to a tradition Design of Experiments (DoE) will be further discussed in the next chapter. The design of experiments would include section of the design space around the selected scenarios.

13.2.5 Analyze the Results and Select the Design Drivers

The analysis of results from the experiments planned in Chapter 14 will be discussed in Chapter 15. The results will include examination of model behavior, relative importance of control and noise variables and interactions, derived from the computer experiment [62]. Sensitivity analysis is important because there is uncertainty in the computer model [111]. Response Surface Methodology (RSM) models will be built to provide insight into trade-offs and sensitivities.

The results will be analyzed for the full coverage of the design space option and the feasible space option. The differences in the results will also lend insight into the number of scenarios that must be included in robust analysis. If the feasible space results approach or correspond to the full coverage results, it can be deduced that it is not necessary to exclude infeasible regions.

CHAPTER XIV

FEASIBLE SPACE DEFINITION

This chapter discusses different design space samplings that will be used in the robust design process. The experiments to run will be developed in this chapter and the results compared in the next chapter.

14.1 Theory of Feasible Space

The concept of feasible space arises from the inclusion of scenarios that are not possible when considering the scenario variables as fully parametric. The full coverage of the design space equally samples all regions of the noise space. Feasible coverage excludes the regions that do not represent realistic scenarios. Some literature exists on the treatment of multiple scenarios in robust design. Vommi and Seetala [192] suggest using a weighted robustness factor, proportional to the likelihood of the scenario. The weighted expected loss (η) is expressed as a function of the normalized frequency (w_j) and the expected loss for that scenario:

$$\eta = \sum_j w_j E(Loss)_j$$

To perform the robust design across the scenarios, η is minimized.

Chapter 6 defined four scenario but there is uncertainty in these scenario. The amount of supplies needed for a humanitarian mission is not known and even the number of units that compose a Marine Expeditionary Brigade are in flux. It is necessary to include this uncertainty in the robust design process. The feasible design space is defined as the segments of the design space covered by the uncertainty around selected scenarios.

14.2 Experimental Designs

It is of interest to see how changes in the amount of the design space sampled impact the sensitivity analysis and key performance parameters. To examine this impact, four sets of sampling will be run and the results compared in the next chapter. These experiments are:

1. Single scenario
2. Discrete scenarios
3. Full coverage of noise space
4. Feasible design space

The relative amount of noise space covered by these experiments are seen in Figure 66. Case 1 design around the blue star. Case 2 covers the entire gray box. Case 3 covers the black dots in addition to the blue star. Case 4 covers the tan blocks, selecting space that is close to the scenarios, but includes uncertainty in the exact requirements of that scenario.

The next four sections will present the generation of the experiments. This will be used in addition to the 19 design variable listed in Table 25.

14.2.1 Control Factors Design of Experiments

Kleijnen et al., [102] reviewed several types of designs for computer experiments and recommends the use of a latin hypercube for problems with many factors and where assumptions of the result are not desired. The design needed in this thesis has many factors that may experience non-linear behavior. Latin hypercube designs have many desirable features that are attractive, but falls short in orthogonality and space-filling criteria [41]. These shortcoming were addressed with the development of nearly orthogonal latin hypercube (NOLH) designs. These designs include space filling criteria and have a maximum correlation between any two columns [116].

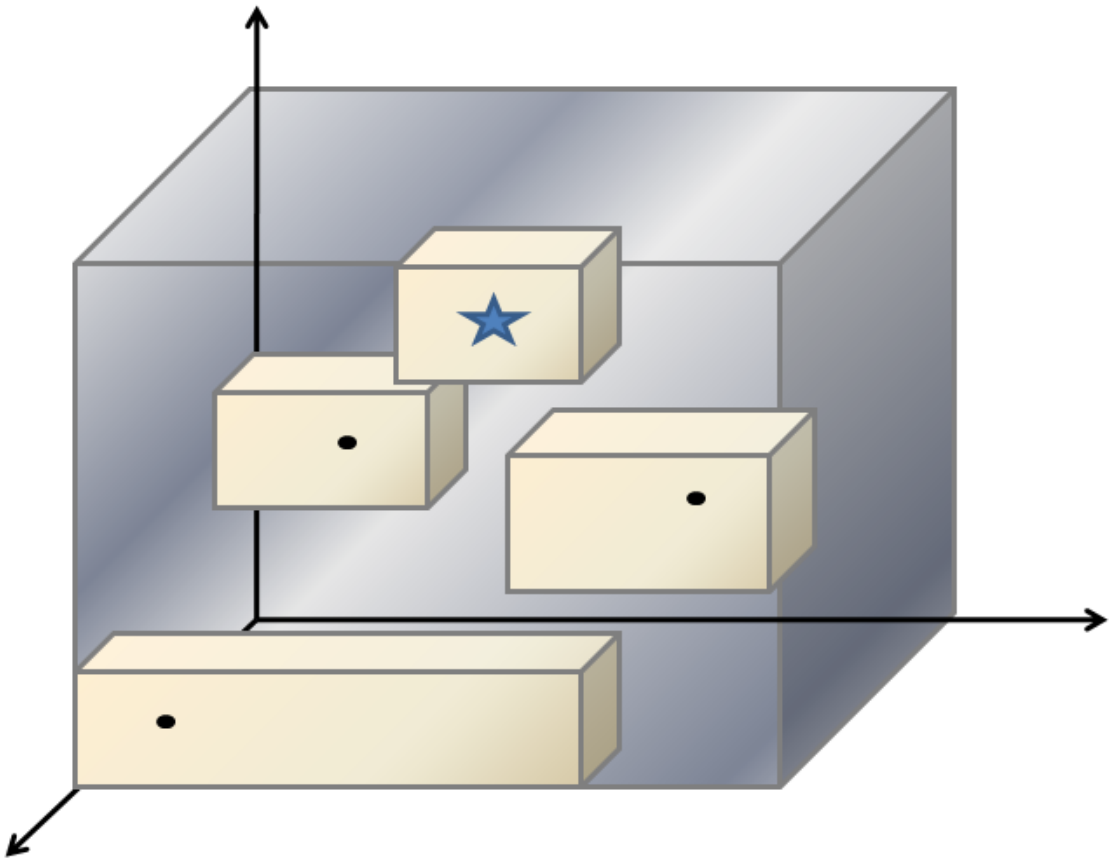


Figure 66: Design Space Coverage

Cioppa et al., [63] describes an algorithm for developing nearly NOLH designs with pairwise correlation less than 0.03 and maximizing space-filling properties. This process was executed for 7, 11, 16, 22, and 29 factor and is available online [159], but the process hold true for larger numbers of variables. For designs of less variables, columns can be removed without loss of space-filling or orthogonality. The design for 22 variables will be used as the basic DoE for the design variables. If more cases are needed, the columns of this design can be rotated without compromising the orthogonality or space-filling properties.

14.2.2 Single Scenario

The robust design using a single scenario is just a consideration of the 19 design variables. Traditionally, Navy vessels are designed for major combat operations, so the large scale military operation scenario is used for this experiment set. The DoE for the design variables is run, requiring 129 cases.

14.2.3 Discrete Scenarios

To measure the performance across the four discrete scenarios, the design DoE is repeated for each of the scenarios. The input values used for each scenario are listed in Table 27.

14.2.4 Full Coverage

The inclusion of the entire noise design space could be done by creating a combined DoE or crossing the design variable DoE with a DoE for the noise variables. Combined DoEs complicate the calculation of the variance for use in the loss function and is suggested only for cases where the runs are very costly [162]. In this cases, the runs are not costly so the DoEs will be crossed for ease of calculation.

A DoE is needed for the 32 noise variables, which exceeds the developed NOLH designs. An algorithm that develops NOLH designs is available through the SEED

Table 27: Scenario Based Noise Variable Settings

Variable	MCO	MEU	Humanitarian	Sustainment
Distance to ISB (nmi)	3000	3000	1000	3000
Sea Base distance (nmi)	200	200	25	100
Number of landing spots	3	3	2	5
Number of austere ports	0	0	1	0
Other vessels present:				
LMSR	3	1	2	2
LHD	11	3	0	0
LSD	11	3	0	0
LPD	11	3	0	0
MLP	6	1	1	1
LCACR	36	4	4	4
JHSV	4	1	1	0
MV22	5	1	0	0
CH46	11	6	0	0
CH53	11	6	0	0
Cargo damand:				
Marine Platoon	156	26	0	0
EFV	94	14	0	0
LAV - 25	54	4	0	0
HMMWV	0	12	0	0
M1A2	94	4	0	0
EFSS Element	0	6	0	0
CEB Element	0	6	0	0
Arty Element	42	0	0	0
Mortar Element	24	0	0	0
Antiarmor Element	24	0	0	0
HIMARS Element	0	0	0	0
Pallet	0	0	0	3000
TEU - Water	0	0	116	0
TEU - MRE	0	0	56	0
TEU - beans	0	0	0	0
TEU - shelter	0	0	20	0
TEU - Eng Co	0	0	4	0

center [159] and generates designs based on the number of variables inputted and the ranges. Independent designs can be generated and appended to reduce multicollinearity among the columns. This design is limited to integer values, but the noise variables are either integers or have large ranges, i.e. range values, so this limitation is not an issue. Two independent designs, 64 total cases, were generated using the ranges in Table 26 and the noise space coverage is pictured for a subset of variables in Figure 67. This figure shows the coverage compared to the four scenarios, the MCO highlighted in red and the other three in green.

14.2.5 Feasible Design Space

To test across the feasible space, DoEs are generated for each individual scenario. Latin Hypercube designs were generated for each scenario. The same number of points are run for each scenario as not to bias the testing and any weighing of interest can be applied to the loss function. The ranges used in the generation of these DoEs are listed in Table 28. The coverage of the distances is conplete for each scenario, but the design space coverage for the other variables is shown in Figure 68. The sampling around the scenarios is seen with the discrete scnearios included in red and green points. The empty space represents infeasible regions of the design space and the coverage can be compared to the full coverage seen in Figure 67.

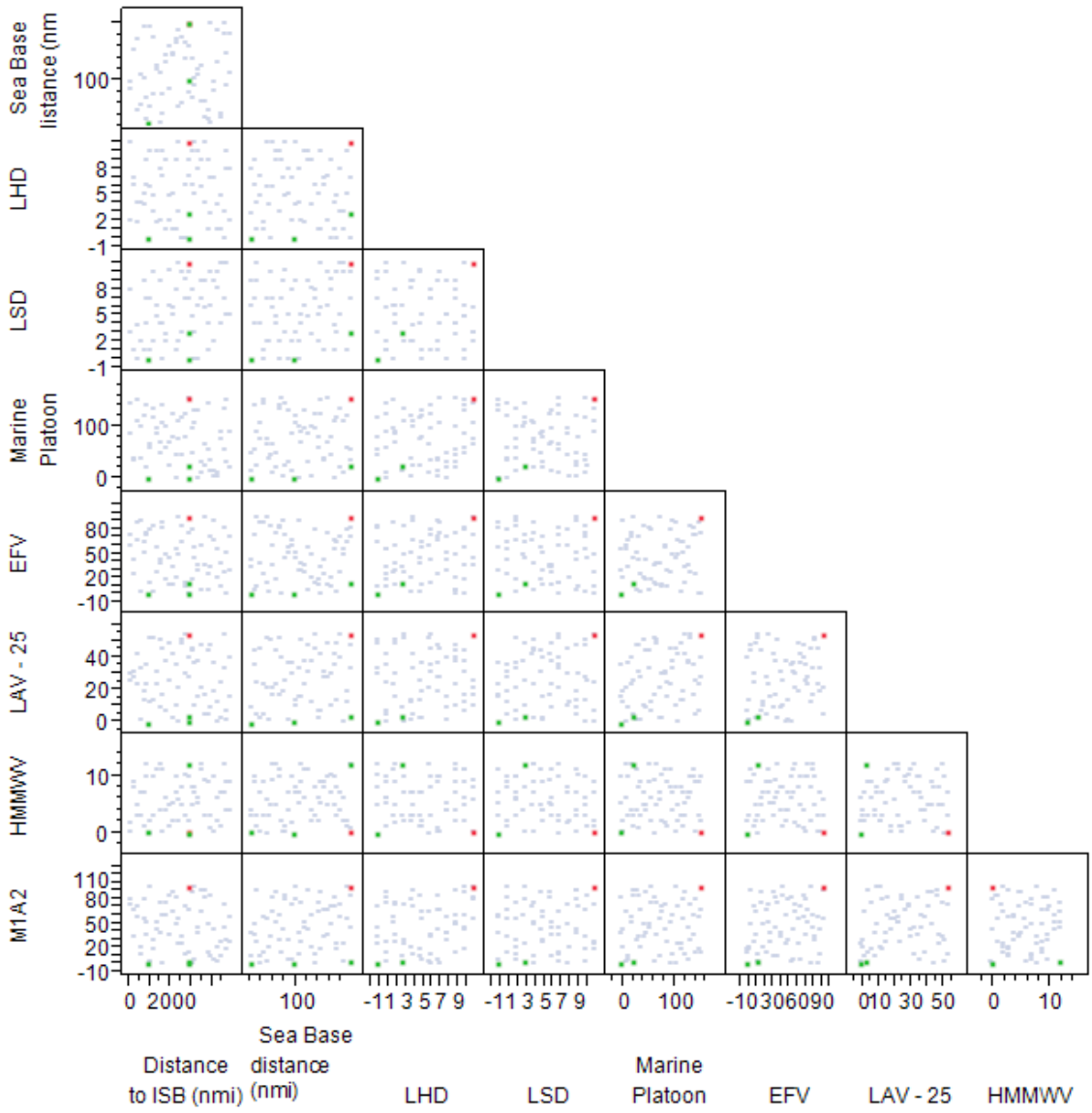


Figure 67: Complete Design Space Coverage

Table 28: Scenario Based Noise Variable Ranges

	MCO		MEU		Humanitarian		Sustainment	
Variable	min	max	min	max	min	max	min	max
Distance to ISB (nmi)	200	5000	200	5000	200	5000	200	5000
Sea Base distance (nmi)	25	200	25	200	25	200	25	200
Number of landing spots	1	5	1	5	1	5	1	5
Number of austere ports	0	2	0	2	0	2	0	2
Other vessels present:								
LMSR	0	3	0	1	1	2	1	3
LHD	7	11	1	3	0	0	0	0
LSD	7	11	1	3	0	0	0	0
LPD	7	11	1	3	0	0	0	0
MLP	3	6	0	1	1	2	0	2
LCACR	12	36	0	4	0	4	0	8
JHSV	2	4	0	1	0	1	0	0
MV22	2	5	0	1	0	0	0	0
CH46	3	11	0	6	0	0	0	0
CH53	3	11	0	6	0	0	0	0
Cargo damand:								
Marine Platoon	100	180	20	30	0	0	0	0
EFV	60	120	10	18	0	0	0	0
LAV - 25	40	60	3	6	0	0	0	0
HMMWV	0	0	8	16	0	0	0	0
M1A2	50	120	3	6	0	0	0	0
EFSS Element	0	0	4	8	0	0	0	0
CEB Element	0	0	4	8	0	0	0	0
Arty Element	30	50	0	0	0	0	0	0
Mortar Element	15	30	0	0	0	0	0	0
Antiarmor Element	15	30	0	0	0	0	0	0
HIMARS Element	0	0	0	0	0	0	0	0
Pallet	0	0	0	0	0	0	1000	3000
TEU - Water	0	0	0	0	100	200	0	0
TEU - MRE	0	0	0	0	20	100	0	0
TEU - beans	0	0	0	0	3	10	0	0
TEU - shelter	0	0	0	0	10	50	0	0
TEU - Eng Co	0	0	0	0	0	6	0	0
Petroleum Pod (500 gal)	500	1000	100	300	0	0	300	1000

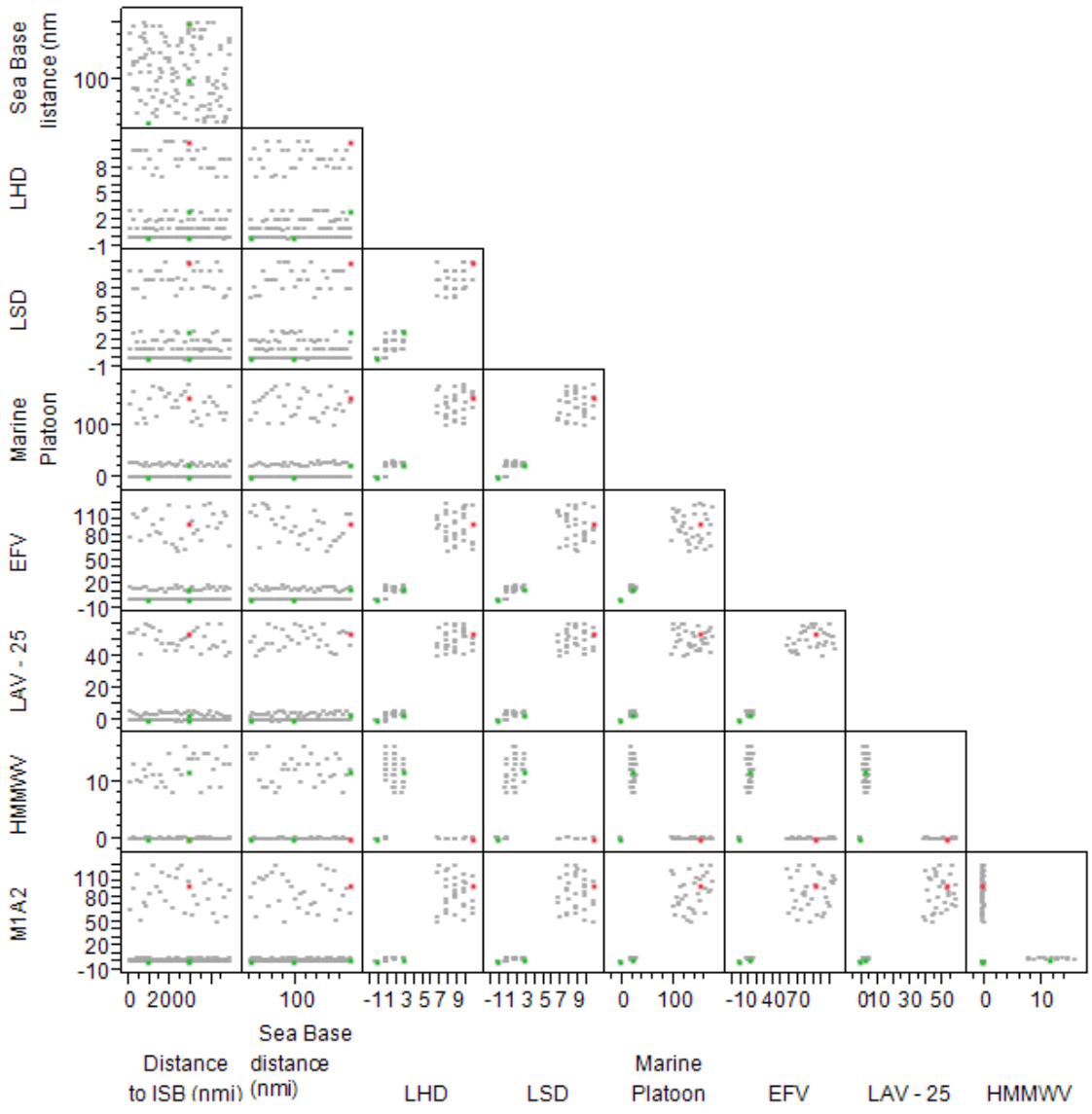


Figure 68: Design Space Coverage

CHAPTER XV

ROBUST DESIGN RESULTS

Taking the four design of experiments discussed in the previous chapter, the model results will be compared. Results for each sampling method will be given, including filtering of the design parameters using a screening test and further analysis through a second order response surface equation. The results are followed by a discussion of the impact of the sampling methods. All of the analysis and graphics were generated using JMP Statistical Software.

15.1 Single Scenario

The first set of results presented are for the major combat operation. The normalized mean of the shortfall and the loss are examined. Since a single scenario is presented, the loss function becomes:

$$Loss = w_1 * \mu^2_{shortfall} + w_2 * \mu^2_{fuelusage}$$

The shortfall with be weighted twice as much as the fuel loss, indicating the performance is more important than that cost. A screening test is performed for both responses and the results are seen in Figure 69 and 70. Incorporating the fuel usages increases the number of design drivers, demonstrating the importance of incorporating a cost metric. Table 29 shows the selected design driver based on the two responses. Since the drivers for shortfall only are also in the factors selected based on loss, the larger number of factors will be carried through for further analysis. It is important to note that if the loss function was modified, the order and magnitude of the design drivers could change.

Term	Orthog Estimate
Number available	-0.1679600
Max lift (LT)	-0.0876764
Max area (sqft)	-0.0367141
Mean time between failure (min)	-0.0188948
SES speed (kts)	0.0164350
Mean time to repair (min)	0.0146417
Beach time to unload (min)	0.0117107
Use Austere Port?	-0.0092942
ACV speed (kts)	0.0091304
Use Stern Ramp?	-0.0079917
Transition distance (nmi)	0.0067119
Time to load - port (min)	-0.0054802
Use Port Ramp?	-0.0051575
Range (nmi)	0.0025249
Transition time (min)	0.0023893
Use Crane?	-0.0010086
Time to unload - austere (min)	-0.0009538
Time to load - crane (min)	-0.0006856
Time to load - stern (min)	-0.0001630

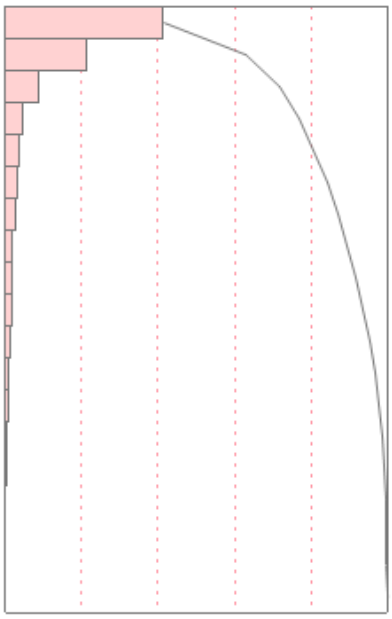


Figure 69: Pareto Plot for Shortfall - MCO

Term	Orthog Estimate
Max lift (LT)	-0.7223054
Max area (sqft)	-0.5318959
Transition time (min)	0.1493983
Mean time between failure (min)	-0.1394828
Number available	-0.1377022
SES speed (kts)	0.1342673
Use Stern Ramp?	-0.1328105
Mean time to repair (min)	0.1011606
Use Port Ramp?	-0.0832338
Beach time to unload (min)	0.0745116
Range (nmi)	0.0735841
ACV speed (kts)	0.0730747
Time to unload - austere (min)	0.0505522
Time to load - port (min)	-0.0446642
Transition distance (nmi)	0.0443846
Time to load - stern (min)	-0.0226977
Use Crane?	-0.0210985
Use Austere Port?	-0.0137017
Time to load - crane (min)	-0.0111235

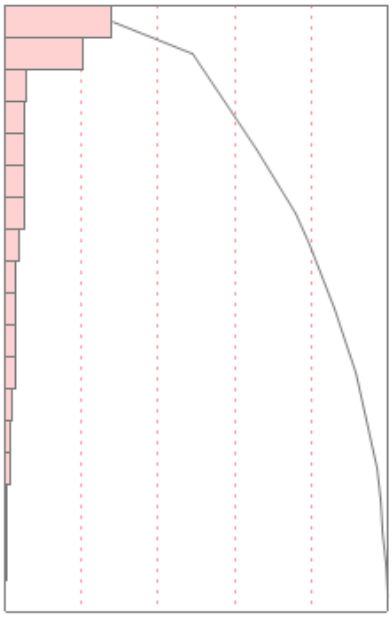


Figure 70: Pareto Plot for Loss - MCO

Table 29: Design Driver Comparison - MCO

Shortfall	Loss
Number available	Max lift (LT)
Max lift (LT)	Max area (sqft)
Max area (sqft)	Transition time (min)
Mean time between failure (min)	Mean time between failure (min)
SES speed (kts)	Number available
Mean time to repair (min)	SES speed (kts)
	Use Stern Ramp?
	Mean time to repair (min)
	Use Port Ramp?

Based on the down-selected design drivers, a second order response surface equation (RSE) is developed. The exception to this second order model are the use of the interfaces because these are are yes-no variables and the second order term does not gain additional information. The prediction of shortfall with this model is good having an R^2 of 0.92. The prediction of loss is not as good with an R^2 of 0.75, but predicts well enough to provide insight into trends. These RSEs are to increase knowledge about the model, not predict, so the shape of the trend is of more importance. There are cases where the predicted value will be less than zero, especially at the edges of the design space, because a subset of the factors are being used, with the other defaulted.

Using a prediction profiler, the sensitivity to variation in the individual parameters are explored, seen in Figure 71. The trends are very similar for shortfall and loss. For best performance, a large number of fast craft are needed. The design factors that directly impact cycle time, such as speeds, transition time, mean time between failures, and mean time to repair are pushed toward their minimal value. It is more interesting that the maximum area has diminishing returns as it is increased and reaches a plateau. The location of this plateau remains the same even with variations in the other design variables. For this scenario, it is the lift that is more important above a certain area threshold. The option to be able to use both the port and stern

ramps are also an important driver. If the number of MECs is decreased below 4, the importance of use of the stern ramp become negligible, with the trend switching direction below three. The trend reversal can be attributed to queuing at the Sea Base.

Figure 72 shows the change of trends at very low number available. Note the flattening of the speed trend as well, with low speeds now decreasing loss and shortfall. This indicates the more ships you have, the faster they should be able to travel. But with a very slow ship that can only use the port ramp, the increased fuel usage actually outweighs the decrease in shortfall and the loss indicates it would be better to have fewer MECs. This increase in fuel usage is because the inputted fuel burns are in gallons per hour so a slower ship takes longer to travel and thus burns more fuel. To increase the accuracy of these trends, a relationship between speed and fuel burn could be incorporated into the DELAS model.

Sizing the MEC for a major combat operation pushes the design of the ship to a large, fast ship. The primary drivers are the lift and area, but the area has diminishing returns. Key second order terms appear above transition time in the Pareto plot, including the number available and the use of the ramps and SES Speed^2. It is important these ships can remain operational for long periods of time and be repaired quickly. The more ships present, the more important it is to have interface options at the Sea Base.

15.2 Discrete Scenarios

The analysis of mean shortfall and loss are repeated for the testing of four discrete scenarios. The results for each of the four scenarios are normalized by the minimum results of that scenario. If not normalized, the magnitude of the shortfall ranges from 200 to 190000 due to the large differences in cargo demanded. If the unnormalized values are used, the loss function is dominated by the variation in shortfall. For the

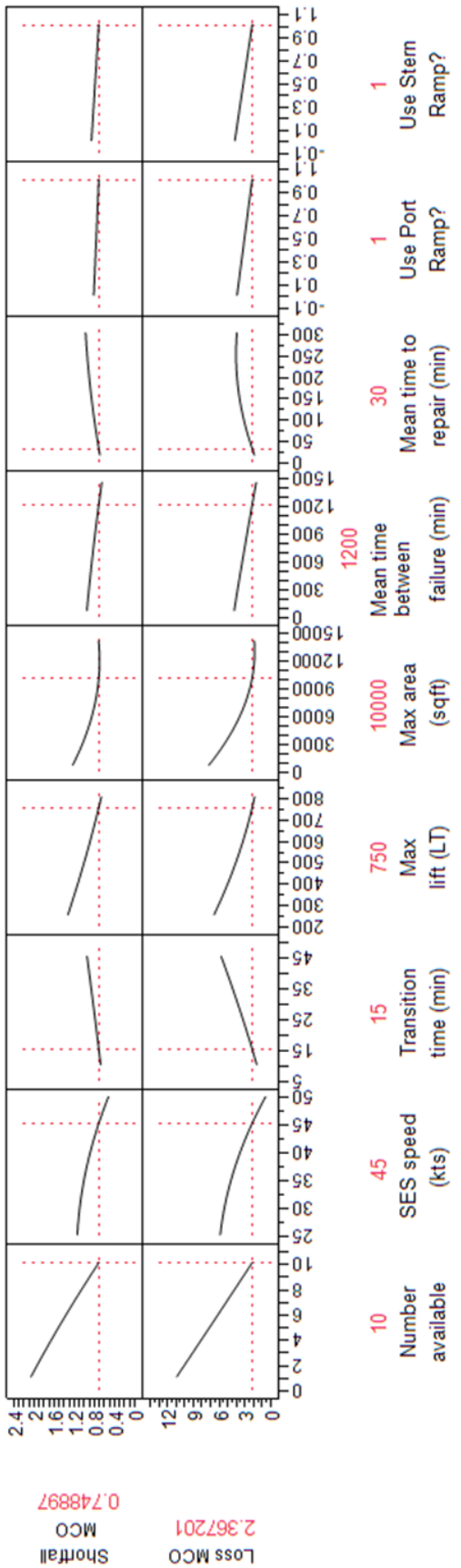


Figure 71: Prediction Profiler - MCO

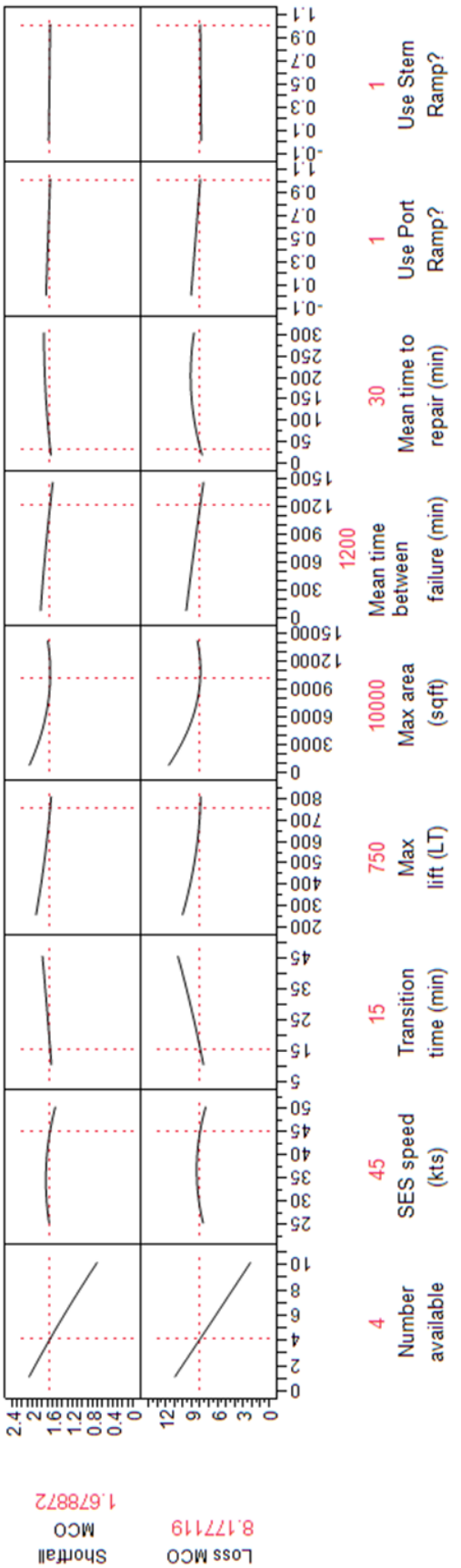


Figure 72: Prediction Profiler 2 - MCO

Table 30: Design Driver Comparison - Scenarios

Shortfall	Loss
Number available	Max area (sqft)
Max area (sqft)	Number available
Max lift (LT)	Max lift (LT)
Beach time to unload (min)	SES speed (kts)
Use Port Ramp?	Use Port Ramp?
SES speed (kts)	Mean time to repair (min)
Mean time to repair (min)	Time to load - port (min)
Range (nmi)	Use Crane?
Use Crane?	Beach time to unload (min)
	Use Stern Ramp?
	ACV speed (kts)

loss function, the shortfall is again weighted twice as much as the fuel:

$$Loss = 2 * (\sigma^2_{shortfall} * \mu^2_{shortfall}) + (\sigma^2_{fuelusage} + \mu^2_{fuelusage})$$

The Pareto plots for the shortfall and loss responses are given in Figure 73 and 74. Screening the two response options results in the drivers in Table 30. Many of the same factors are identified as with the major combat operation, including number available and life and area of the ship. But factors such as beach time to unload and time to load using the port ramp have emerged as important. The most interesting drop is the mean time between failures, indicating when shorter operations are considered the reliability is not as important. Transition time has also dropped significantly indicating it is no longer a driving component of the total travel time.

The major difference between the two responses screening tests are the incorporation of range. Since it is fairly low on the shortfall screening test, the same second order model variables will be used for both responses, including the variables listed for loss. This model provides a fairly good estimation of both responses with an R^2 of 0.83 for shortfall and 0.84 for loss.

Figure 75 shows many similar trends to the analysis of just the MCO, such as wanting many MECs and having a diminishing return on increasing area. The plateau

Term	Orthog Estimate
Number available	-0.1114926
Max area (sqft)	-0.1031834
Max lift (LT)	-0.0573795
Beach time to unload (min)	0.0465373
Use Port Ramp?	-0.0367668
SES speed (kts)	-0.0358235
Mean time to repair (min)	0.0281537
Range (nmi)	-0.0198798
Use Crane?	0.0157741
Time to load - port (min)	-0.0144554
Time to load - crane (min)	-0.0139617
Transition distance (nmi)	0.0129572
Time to unload - austere (min)	-0.0121461
Use Austere Port?	-0.0101266
ACV speed (kts)	0.0097960
Time to load - stern (min)	0.0097096
Use Stern Ramp?	0.0094007
Transition time (min)	-0.0076548
Mean time between failure (min)	0.0044984

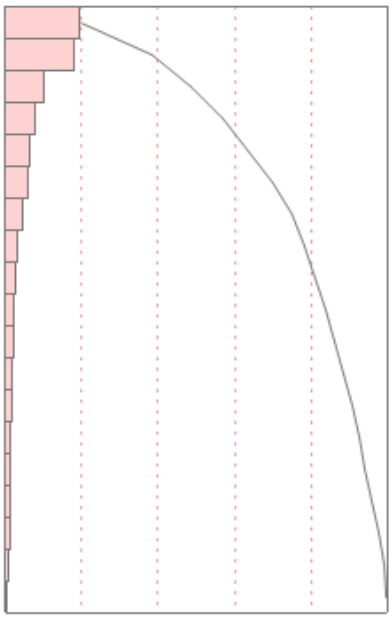


Figure 73: Pareto Plot for Shortfall - Scenarios

Term	Orthog Estimate
Max area (sqft)	-3.751492
Number available	2.000073
Max lift (LT)	-1.173544
SES speed (kts)	-0.885036
Use Port Ramp?	-0.711607
Mean time to repair (min)	0.622793
Time to load - port (min)	-0.557237
Use Crane?	0.463724
Beach time to unload (min)	0.425222
Use Stern Ramp?	0.402356
ACV speed (kts)	0.286581
Use Austere Port?	0.199721
Time to unload - austere (min)	-0.182299
Transition distance (nmi)	0.170472
Mean time between failure (min)	-0.165641
Transition time (min)	-0.164919
Time to load - stern (min)	-0.157238
Time to load - crane (min)	-0.139802
Range (nmi)	-0.094503

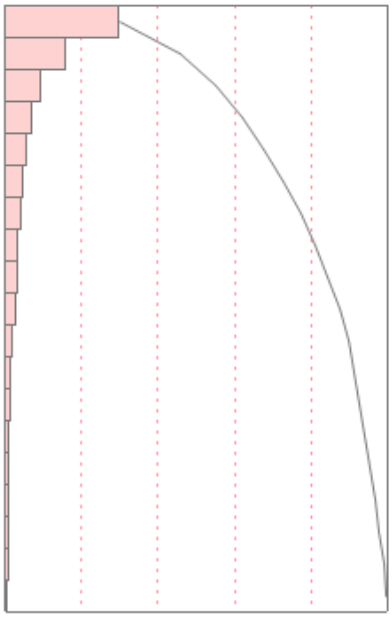


Figure 74: Pareto Plot for Loss - Scenarios

in area occurs with a smaller ship for shortfall than loss. The curvature of the speed is an over fit and should be flat, indicating that if enough ships are preset, the speed is not important. Figure 76 show the increasing importance of speed as the number of ships is decreased.

A trade-off is present between mean time to repair and beach time to unload. If it takes more than 5 hours to unload the connector, then it is desirable to have a short mean time to repair, but with a quick unload time, a longer mean time to repair is better. This indicates that the bottle neck occurs at the Sea Base so it is desirable to space out the returning connectors. Yet, a greater load time using the port ramp is desirable, plateauing at 7.5 hours. This trend is related to the maximum lift. If it is a small ship, a quick load time is needed but with a larger ship, a larger load time is fine. With a larger ship, less trips are needed to less queuing would occur. Although having load options is good, including a crane option is not, due to the range for the load time using the crane being larger than for the other two load options. Since the model is built to use the fastest available interface, this shows that it is better to wait for a quicker interface then to use the slower crane.

The incorporation of more scenarios into the loss analysis has raised an important trade-off between speed and number available. There is a choice here to have more ships or fewer fast ships. But this is coupled with the impact of increasing lift, which is of greater importance with fewer ships. The importance of interface option is demonstrated but also highlights the negative impact of a significantly slower option. The speed of the interface is more closely coupled with the maximum lift than the number available, a surprising result in that the total number of trips needed is more important than the queuing at the Sea Base.

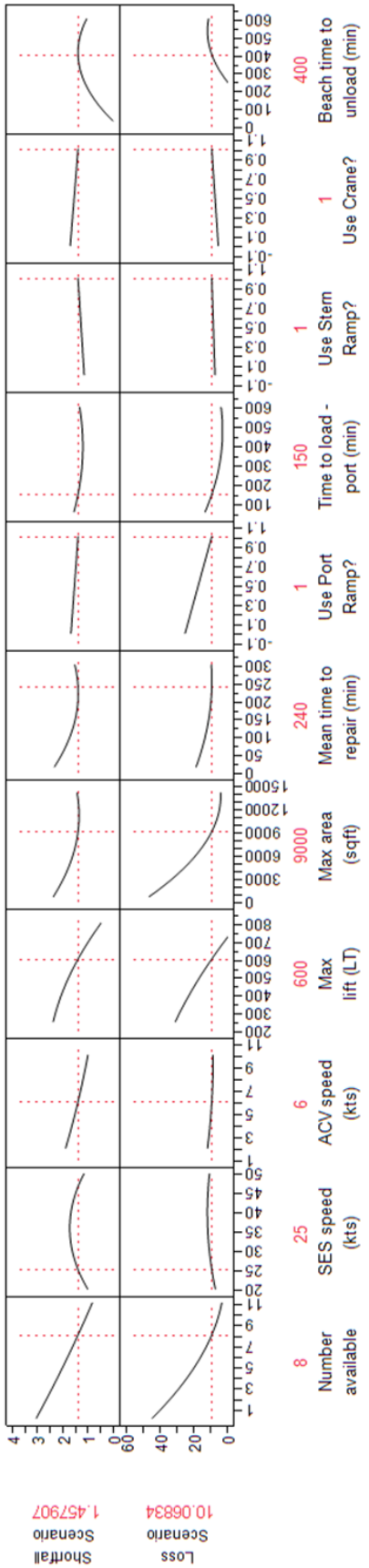


Figure 75: Prediction Profiler - Scenarios

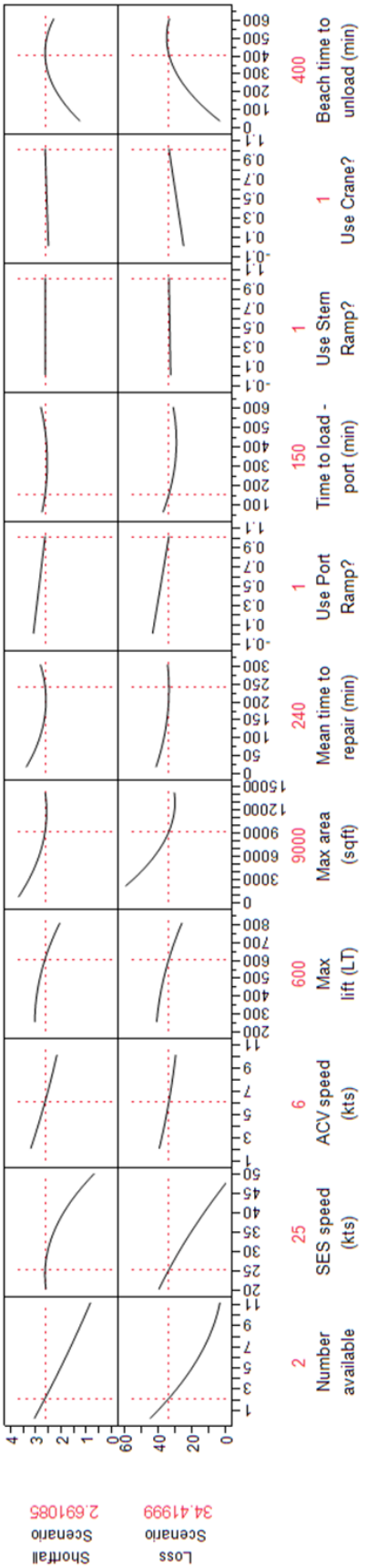


Figure 76: Prediction Profiler 2 - Scenarios

15.3 Full Coverage

This set of results represents the first where the noise space is sampled beyond a small set of scenarios. The DoE generated for the noise space covers the entire ranges. Once again, the shortfall is weighted twice as much as the fuel usage in generating the loss function. Looking at the individual results, the variation in the performance metric is not great, but the difference in fuel usage varies greatly. Since the values are normalized within each case, this large variation can not be attributed to differences in the scenario, instead it is clear that different MEC designs and numbers cause great differences in how the response operation is formulated and the usage of assets.

The Pareto plots for shortfall and loss are given in Figure 77 and 78. The design drivers are listed in Table 31 and the drivers for shortfall and loss vary for the first time. Main drivers do appear again, such as the number available and SES speed. The drivers for shortfall remain similar, including the lift and area capabilities of the ship, interface options and repair criteria. It is interesting that the lift and area have dropped significantly for the loss. As expected, fuel variation appears to be driving the loss with the emergence of SES speed and range. If the range is not large enough for the MEC to travel from the ISB to the destination, the operation would have to rely on other connectors. This fuel variation could also explain the emergence of the load and unload time as the longer an MEC has to queue, the more fuel it burns.

Since the design drivers varied, two separate RSEs will be developed incorporating different variables. The mean shortfall model has a very good fit with an R^2 of 0.96. Figure 79 shows the prediction profiler for shortfall, which again trends toward many, large ships. Again a plateau is seen in the max area, occurring near 12000 sqft. The speed is not as important, again indicating that having a large number of ships makes the speed not as important. With a slow ship, the penalty for increasing beach time to unload is not as great. This highlights the relationship between the speed of the ship and penalty for load or unload time as additional advantage is not gained with

Term	Orthog Estimate
Number available	-0.2389465
Max lift (LT)	-0.1018844
Max area (sqft)	-0.0900875
SES speed (kts)	0.0683903
Use Port Ramp?	-0.0530586
Mean time to repair (min)	0.0308205
Beach time to unload (min)	0.0286108
Use Austere Port?	-0.0251586
Mean time between failure (min)	-0.0243834
Time to unload - austere (min)	-0.0107530
Transition time (min)	-0.0067062
Time to load - stern (min)	-0.0066802
Time to load - port (min)	0.0056890
Transition distance (nmi)	0.0056069
Use Stern Ramp?	-0.0048730
Use Crane?	-0.0046806
Time to load - crane (min)	-0.0024032
ACV speed (kts)	-0.0019433
Range (nmi)	0.0001441

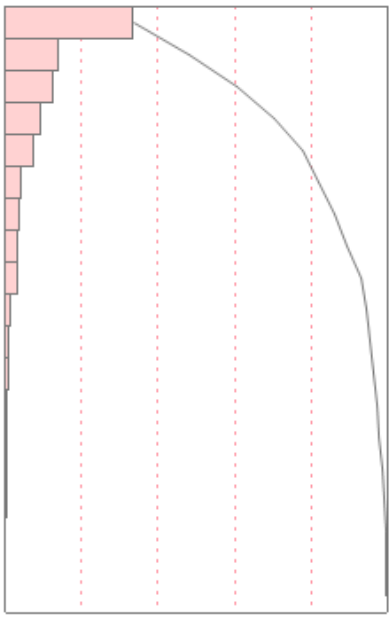


Figure 77: Pareto Plot for Shortfall - Full Space

Term	Orthog Estimate
Number available	5474.099
SES speed (kts)	913.818
Range (nmi)	806.664
Time to load - port (min)	508.453
Beach time to unload (min)	-408.688
Time to unload - austere (min)	296.742
Time to load - crane (min)	-275.472
Max area (sqft)	-237.971
Use Port Ramp?	-135.430
Max lift (LT)	-134.805
Transition time (min)	117.311
Use Stern Ramp?	104.972
Use Crane?	73.121
ACV speed (kts)	-71.768
Transition distance (nmi)	32.449
Time to load - stern (min)	19.955
Mean time to repair (min)	-16.063
Use Austere Port?	-15.896
Mean time between failure (min)	10.553

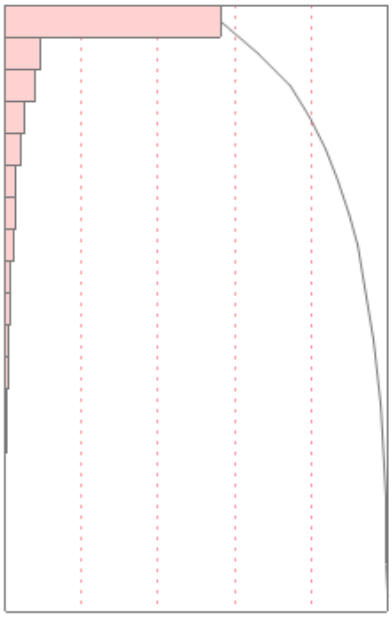


Figure 78: Pareto Plot for Loss - Full Space

Table 31: Design Driver Comparison - Full Space

Shortfall	Loss
Number available	Number available
Max lift (LT)	SES speed (kts)
Max area (sqft)	Range (nmi)
SES speed (kts)	Time to load - port (min)
Use Port Ramp?	Beach time to unload (min)
Mean time to repair (min)	Time to unload - austere (min)
Beach time to unload (min)	Time to load - crane (min)
Use Austere Port?	Max area (sqft)
Mean time between failure (min)	

a fast ship if the time gained in arriving at the beach faster is spent queuing for the beach.

The RSE for loss has a good fit with an R^2 of 0.90 and the prediction profiler in Figure 80. The trends demonstrated here are clearly driven by the variation in fuel usage. The desirability is pushing toward no MECs. If any are included, they should have a large area and be able to load quickly. The push to no range is related to the number available. If the operation range exceeds the MEC range, then the MEC is not deployed, having the same effect as no MECs present. There are countering trends for unload time, with unloading at the austere port requiring less than 3 hours but the beach unloading being pushed to longer times. The push to longer beach unloading is a reversal between shortfall and loss, with this push resulting from a desire to reduce the variability by penalizing additional MECs by not being able to unload them. Overall, these trends are pushing toward fewer vessels and minimizing queuing.

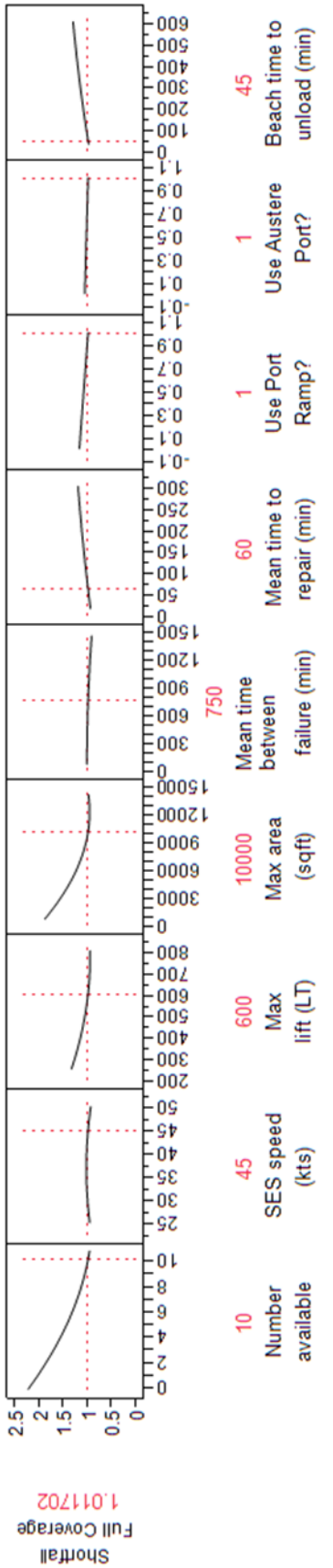


Figure 79: Shortfall Prediction Profiler - Full Space

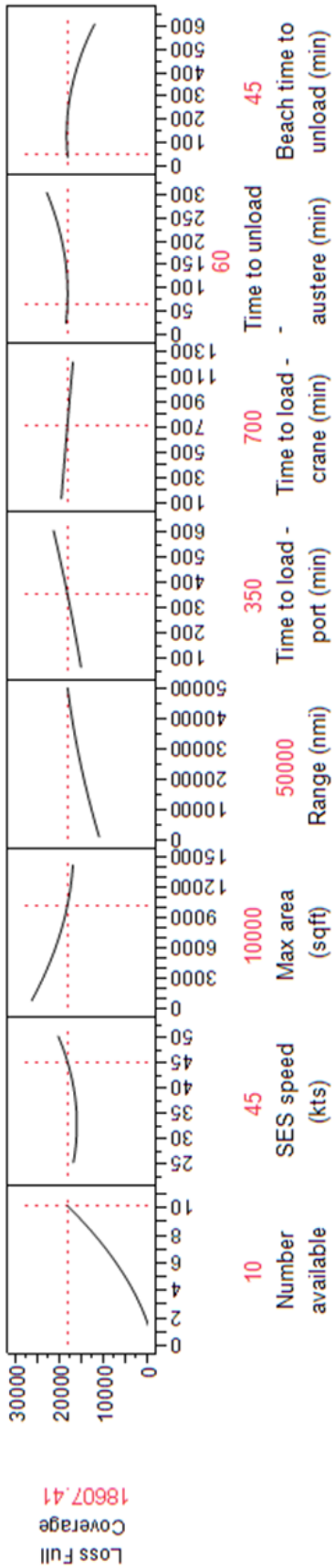


Figure 80: Loss Prediction Profiler - Full Space

15.4 Feasible Design Space

For the feasible design space, the loss functions can be calculated for each scenario DoE, thus it is possible to weigh the loss of each scenario in addition to each response:

$$\eta = \sum_j w_j E(Loss)_j$$

For this study, the four scenarios will be weighted evenly to demonstrate this evaluation concept. Again, the shortfall will be weighted twice as much as the fuel usage. The variance in shortfall and fuel usage is much greater with this sampling, especially in the humanitarian and sustainment operations. The differences in the fuel usage are about five fold greater than the shortfall. The fuel values are similar to those for the full coverage, but the shortfall values are six to ten times greater. The increase in shortfall mean indicates the areas sampled are more challenging to the fleet composition used. The increase in variance indicate the noise parameters are causing a greater variance in the results. The overall larger differences between the designs highlight the potential for a greater improvement with changes in the MEC design.

Figure 81 shows the Pareto for shortfall, which has many of the same drivers seen previously. The max lift has dropped relative to max area, indicating the feasible space is not as weight constrained as the previous DoEs. As with the full coverage, the fuel usage variation is large. The variance in the shortfall has increased, driven by shortfalls near zero that do not normalize well, but remains less than half the fuel usage variation. As seen in Figure 82, the loss is again driven by the number of MECs present, indicating that the loss function is driven more by the variation then the performance.

Using the design drivers listed in Table 32, RSEs were developed. The shortfall RSE has an R^2 of 0.85. Figure 83 and 84 show the coupling between lift and area constraints. A small lift capability causes the maximum area minimum to be much smaller than previously seen and drives the need for more MECs. Increasing the max

Term	Orthog Estimate
Beach time to unload (min)	5.375532
Number available	-2.773743
Max area (sqft)	-2.682651
Transition distance (nmi)	1.637340
Mean time to repair (min)	1.626437
SES speed (kts)	-1.385143
ACV speed (kts)	-1.245450
Time to load - crane (min)	0.916310
Max lift (LT)	-0.669839
Time to load - port (min)	-0.654694
Range (nmi)	-0.538144
Time to load - stern (min)	0.533128
Use Crane?	0.344263
Transition time (min)	0.187159
Time to unload - austere (min)	-0.163543
Mean time between failure (min)	0.142960
Use Austere Port?	-0.123566
Use Stern Ramp?	-0.113792
Use Port Ramp?	-0.057865

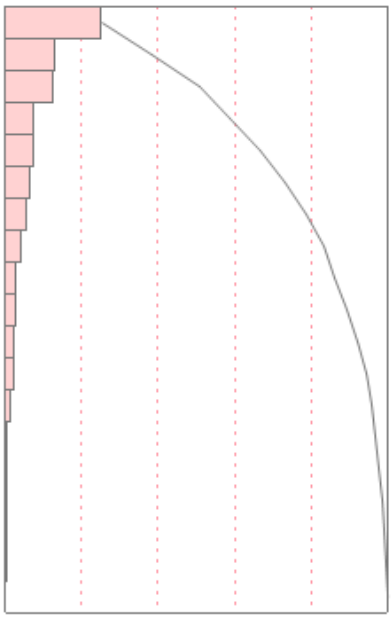


Figure 81: Pareto Plot for Shortfall - Feasible Space

Term	Orthog Estimate
Number available	1499800
Max area (sqft)	-246685
Max lift (LT)	-225267
Range (nmi)	212942
Time to load - port (min)	177750
Use Austere Port?	-137902
Mean time to repair (min)	-122711
SES speed (kts)	121707
Beach time to unload (min)	-105470
ACV speed (kts)	-66032
Use Crane?	-61432
Time to unload - austere (min)	32140
Transition time (min)	26269
Use Port Ramp?	-24734
Use Stern Ramp?	24460
Time to load - crane (min)	-9534
Transition distance (nmi)	6994
Time to load - stern (min)	4747
Mean time between failure (min)	-2957

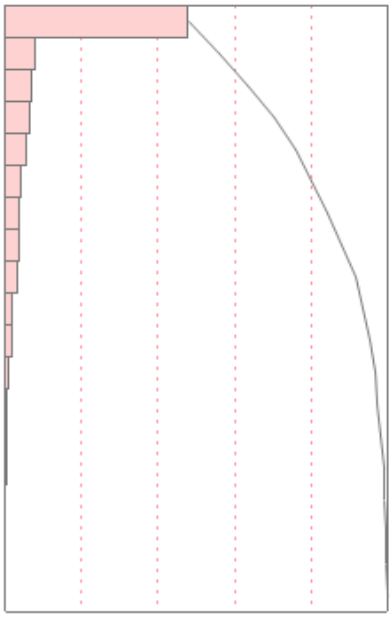


Figure 82: Pareto Plot for Loss - Feasible Space

Table 32: Design Driver Comparison - Feasible Space

Shortfall	Loss
Beach time to unload (min)	Number available
Number available	Max area (sqft)
Max area (sqft)	Max lift (LT)
Transition distance (nmi)	Range (nmi)
Mean time to repair (min)	Time to load - port (min)
SES speed (kts)	Use Austere Port?
ACV speed (kts)	Mean time to repair (min)
Time to load - crane (min)	SES speed (kts)
Max lift (LT)	Beach time to unload (min)
Time to load - port (min)	

lift moves the area minimum, with the increase at large areas due to over fit of the 2nd order RSE. The larger, heavier lift ship reduces the need for additional MECs. The sampling for the feasible space is focused more on the lighter scenarios - where the total weight and area are not great and if the ship is large enough, the operation can be completed with one or two MECs. Adding additional MECs does not improve the performance, as was seen in Section 12.5. The trend reversal present for time to load - port is surprising. If the MEC is small or the beach load time is large, there is no impact or a negative impact in shortening time to load, but with a larger ship that can be unloaded faster, the load time should be reduced below five hours.

The RSE for the loss has an R^2 of 0.92 and results in the trends shown in Figure 85. Once again, the loss pushes the results to fewer ships, driven by the variation more than the performance. The increased impact of the lighter missions pushes toward a larger ships. The larger ship reduces the variability cause by having more MECs. This variability is further reduced by requiring a long time to unload at the beach. Overall, the loss is reduced by design variables that reduce the number of MEC trips required or able to be completed. This trend is also seen in the speed, with the speed minimum location increasing as the other variables are tuned to minimize loss. The option to use the austere port decreases the loss and is more important as the beach

unload time increases, due to reducing the variability in the shortfall by reducing beach queuing.

15.5 Selection of Design Drivers

This chapter presented the results for four separate methods of sampling the noise space in a robust design process, ranging from one scenario to full experimentation in the noise space. Considering the performance, captured in the shortfall metric, many design drivers remained the same over all of the testing methods. The number available, maximum lift, maximum area, and SES speed were consistently design drivers. There was a trade-off in the number available and size along with speed. The difference in the trade-off between full coverage and feasible coverage of the design space is illustrated in the difference between Figure 86 and 87. The shortfall contours are in red and for the full coverage are almost horizontal, indicating a need for more MECs, where in the feasible coverage, those ships also need to be faster. The blue contours are overall loss, which push toward fewer ships with the feasible space and fewer, slower ships for full coverage.

The secondary design impacts come from factors that directly impacted the time per trip, such as the time between repairs and time to repair. As the noise sampling moved from four scenario to full coverage to feasible space, the option to use interfaces were replaced with the time to load at these locations and the time to unload at the beach gained importance. The change in impact can be attributed to the reduction in the number of needed trips with the feasible space. The four scenarios were at the higher demand than the feasible space, so more trips would be needed for those than most of the feasible space cases, leading to loading options being more important. The four scenarios also had a medium value for unloading spots at the beach and the variation in the number of unloading spots made the loading and unloading times more important.

Figure 83: Shortfall Prediction Profiler - Feasible Space

Figure 84: Shortfall Prediction Profiler 2 - Feasible Space

Figure 85: Loss Prediction Profiler - Feasible Space

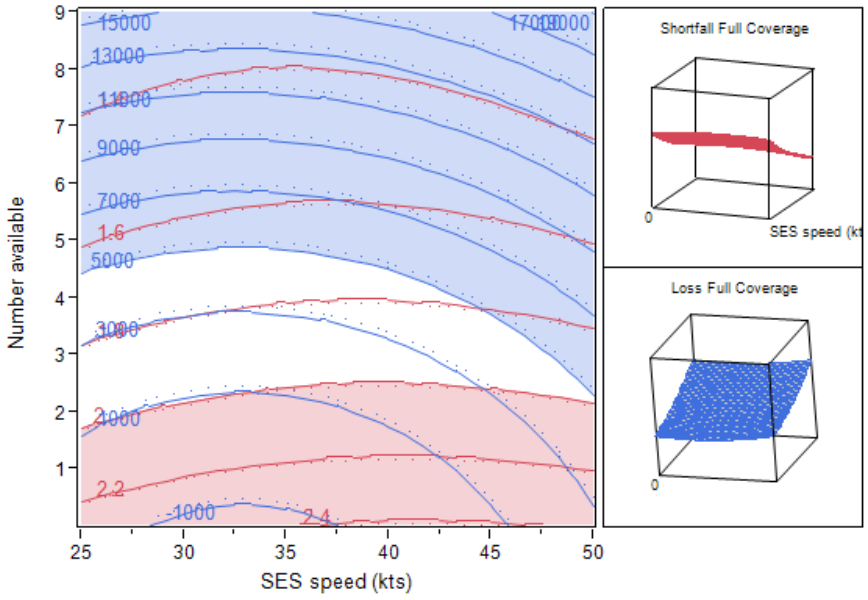


Figure 86: Contour Profiler - Full Coverage

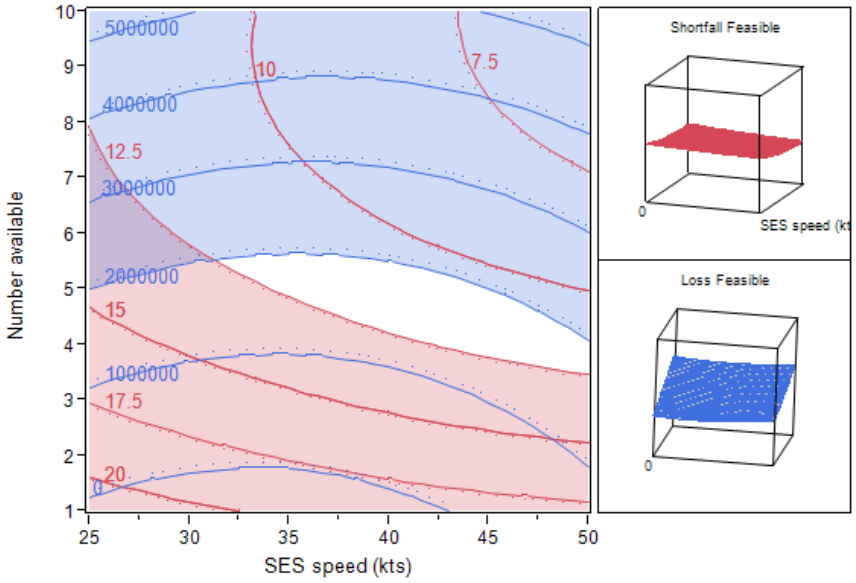


Figure 87: Contour Profiler - Feasible Coverage

The drivers of loss varied greatly through the four sampling methods. This can be attributed to the increased amount of variation. In the initial sampling, there is not variation since only one case is considered, but with the feasible space, the variability was large. The increased variation is mostly driven by the number of MECs present, which is a logical driver. The attributes of the MEC do no create as much variability as the number present. In reality, the variation is not as important as the performance across the scenarios. In cases where there is large variation within a design case, loss as it has been presented in this work may not be the best selection criteria.

15.5.1 Additional Analysis

Due to the loss function being dominated by the variability, additional analysis will be performed considering only the result values of the shortfall and fuel usage. An additional goal of this section is to see how the noise impact compare to those of the design drivers. The full coverage of the noise space and the feasible space sampling will be compared throughout.

Figures 88 through 91 are sensitivities for all design and noise variables. The first two figures are for full coverage and the second two for feasible space. For shortfall, in both cases, the first factor is the number of pallets, which is a recurring demand and shapes the shortfall curve after the initial demand. For full coverage, landing spots follows number of M1A2 units, but is closely followed by the other heavy and large cargo items. This is very different from the feasible space results, which is more driven by the other assets present, particularly the LMSR and other connectors. Here the number available is more important than the number of landing spots. The relative importance of the number of landing spots highlights this variable should be moved to a design choice in setting the climb angle capabilities of the MEC.

Comparing full and feasible space results, the feasible space reduces the coverage of the noise variables, especially the cargo requirements, resulting in a decrease in their

relative importance. In the case of full coverage, the SB distance is a larger driver than the ISB distance, which reverses for feasible coverage. This can be attributed to fewer return trips to the Sea Base for additional cargo making that distance less important. Looking at just the design criteria, by moving to just the feasible space, the top five design drivers do not change, but the options to use interfaces becomes more important than the times to load.

Looking at the fuel usage drivers, the reduction of the design space had a large impact. For the full coverage, the first driver is number of MECs available, followed by the major connectors. A few of the cargo units are interspersed with design choices such as range and time to unload at the beach. The feasible space sampling has the number of LHDs presents as the main driver, followed by the number of pallets. This is followed by the number available and distance to the ISB, highlighting that the large fuel burn for these cases are on the ISB legs of the trip.

The feasible space tests cut down on many of the large, heavy cargo items, translating to fewer trips needed for initial delivery. The other cargo units and ships present are intermixed along with the lift and area of the MEC. The size of the MEC is more important then the presence of some other vessels. Queuing does not seem to be a driver as the interface options and times to load and unload are of lower importance. The reduction to feasible space reduced the impact of cargo units because the reduced space largely occurred at higher demands for large, heavy objects, such as M1A2 units. In terms of the design factors, increasing the number available increases the fuel usage, but increasing the max lift and area reduces it. The higher demand present in the full coverage creates more queuing as more trips are needed, so this queuing increases fuel usage. With the feasible cases, higher speed is more important because it can decrease fuel usage, since fuel usage is based on time.

Term	Orthog Estimate
Pallet	79571.32
M1A2	45844.88
Number of landing spots	-40452.24
Antiarmor Element	29562.05
HMMWV	28364.23
Marine Platoon	-24782.80
EFSS Element	-24771.24
Number available	-21585.15
CH53	-21464.50
LSD	-20133.47
MLP	15426.43
SES speed (kts)	13462.77
Petroleum Pod (500 gal)	12731.16
Number of austere ports	-12099.98
LPD	11190.68
JHSV	-9968.87
Mortar Element	9874.18
TEU - MRE	-9558.68
Max lift (LT)	-9549.10
TEU - Water	8845.38
CH46	-8665.36
Max area (sqft)	-7864.83
LHD	7716.19
LCACR	-7033.25
TEU - shelter	6013.12
CEB Element	-5819.32
Sea Base distance (nmi)	5695.31
Use Port Ramp?	-4585.23
EFV	-3736.44
LAV - 25	3640.47
Mean time to repair (min)	2329.04
Mean time between failure (min)	-2280.00
HIMARS Element	2209.70
TEU - Eng Co	-1808.01
Use Austere Port?	-1762.33
Distance to ISB (nmi)	-1561.91
TEU - beans	1037.88
Time to load - port (min)	933.67
Range (nmi)	802.09
Use Stern Ramp?	-739.27
Beach time to unload (min)	730.01
MV22	-662.76
Time to load - stern (min)	-440.29
ACV speed (kts)	396.92
Use Crane?	-386.58
Time to unload (min)	-373.60
Transition time (min)	-366.87
Transition distance (nmi)	343.25
LMSR	-300.03
Arty Element	242.60
Time to load - crane (min)	-235.67

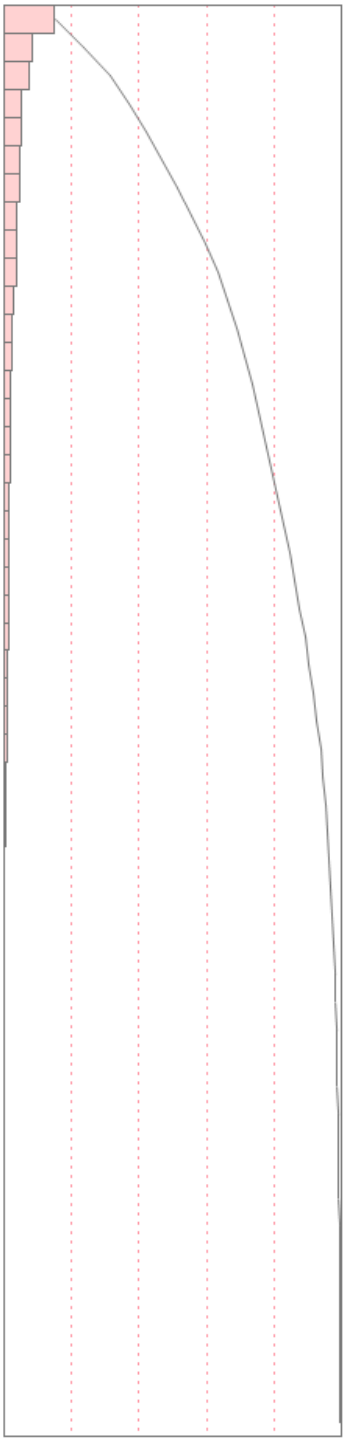


Figure 88: Pareto Plot for Shortfall - Full Coverage, All Variables

Term	Orthog Estimate
Number available	13287623
MLP	9314214
LHD	7453601
CH53	4830678
HMMWV	-3275928
TEU - shelter	2818496
M1A2	2761959
TEU - Eng Co	-2687302
LPD	2682976
HIMARS Element	-2405553
EFSS Element	-2284329
TEU - beans	-2231799
TEU - Water	-2151043
Max lift (LT)	-1937671
LSD	1923107
Max area (sqft)	-1873943
Marine Platoon	-1778224
EFV	1628806
Pallet	1348432
Use Port Ramp?	-1202344
MV22	1185180
LCACR	-1182155
Mortar Element	-1063284
TEU - MRE	1054284
Number of landing spots	856445
Antiarmor Element	803083
Arty Element	-795203
Range (nmi)	726373
Beach time to unload (min)	-689177
Transition distance (nmi)	667984
Number of austere ports	655911
Mean time between failure (min)	-574816
Petroleum Pod (500 gal)	534366
CEB Element	-497222
Sea Base distance (nmi)	-425731
Use Stern Ramp?	-412570
Time to load - port (min)	372739
Distance to ISB (nmi)	-344376
Use Austere Port?	-314250
LMSR	302777
LAV - 25	294651
CH46	293586
ACV speed (kts)	-283780
Transition time (min)	250006
SES speed (kts)	-249412
Time to load - crane (min)	-232393
JHSV	221328
Use Crane?	-184330
Mean time to repair (min)	162863
Time to load - stern (min)	130088
Time to unload (min)	83441

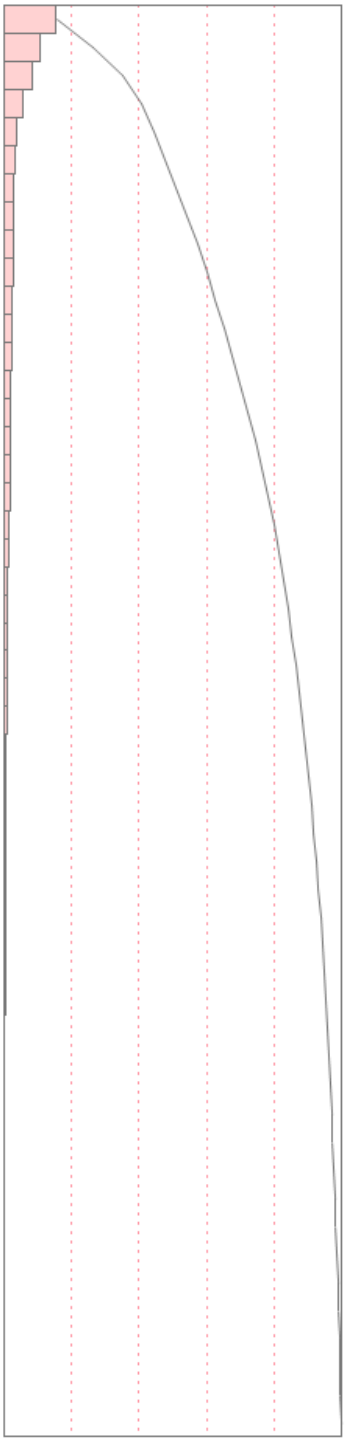


Figure 89: Pareto Plot for Fuel - Full Coverage, All Variables

Term	Orthog Estimate
Pallet	73502.78
LMSR	43539.58
Antiarmor Element	-26410.87
LCACR	22686.20
CH46	-22220.00
HMMWV	-21069.90
LHD	17577.57
JHSV	15956.20
Petroleum Pod (500 gal)	11158.36
Arty Element	-11130.54
Marine Platoon	-10328.65
EFV	-10010.43
MV22	-9859.46
Number available	-9543.41
EFSS Element	-7096.76
MLP	-6730.10
LAV - 25	-6276.82
CEB Element	-6146.72
LPD	5790.23
Number of landing spots	-5679.64
CH53	-5613.78
SES speed (kts)	5324.39
LSD	4961.44
Max lift (LT)	-4388.57
Distance to ISB (nmi)	-4375.15
Max area (sqft)	-3232.60
TEU - Water	-3122.96
Mortar Element	-2839.44
Sea Base distance (nmi)	2243.87
Use Port Ramp?	-1792.97
M1A2	1711.25
Mean time to repair (min)	1203.85
Use Austere Port?	-1190.72
Number of austere ports	-1144.61
TEU - shelter	1132.96
Mean time between failure (min)	-993.09
TEU - beans	-776.87
Beach time to unload (min)	454.41
Time to unload (min)	-344.67
Use Crane?	-325.15
Use Stern Ramp?	-297.33
Transition time (min)	-287.66
Time to load - stern (min)	-287.21
TEU - Eng Co	-240.41
TEU - MRE	-239.01
ACV speed (kts)	215.64
Range (nmi)	214.48
Time to load - port (min)	123.96
Transition distance (nmi)	114.33
Time to load -crane (min)	16.83

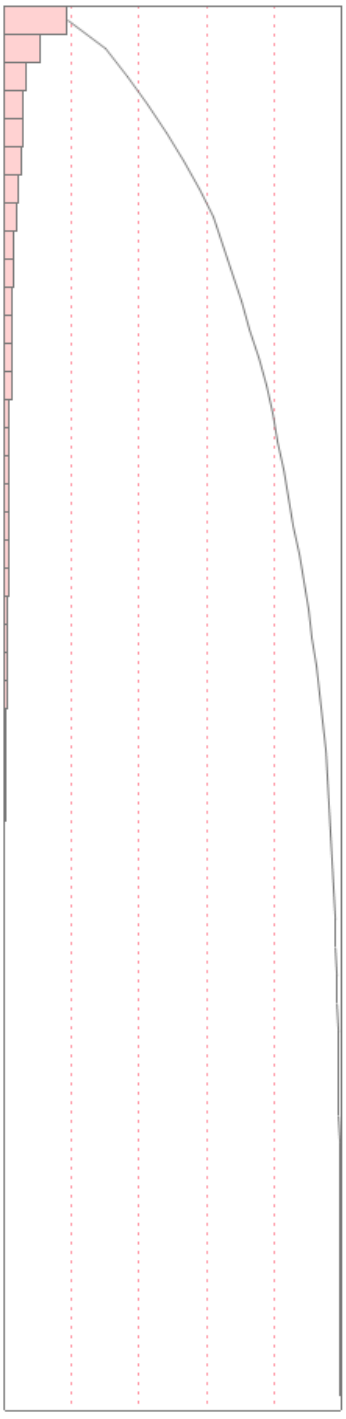


Figure 90: Pareto Plot for Shortfall - Feasible Space, All Variables

Term	Orthog Estimate
LHD	20374565
Pallet	8535366
Number available	6230146
Distance to ISB (nmi)	5944042
Arty Element	-4154070
LMSR	3972412
Antiarmor Element	-3730723
Petroleum Pod (500 gal)	3671748
TEU - Water	-3429811
CH46	-3321515
MLP	3158904
LSD	3036975
Max area (sqft)	-2498428
Marine Platoon	-2465301
LPD	2042841
HMMWV	-1876669
Mortar Element	1579765
Max lift (LT)	-1538294
JHSV	1378520
TEU - MRE	-1241067
EFV	-1194987
TEU - beans	-966975
CH53	911300
Use Port Ramp?	-652298
Number of austere ports	-594060
Range (nmi)	510368
MV22	-496791
Mean time between failure (min)	-465643
M1A2	428166
Transition distance (nmi)	416442
Sea Base distance (nmi)	-382438
SES speed (kts)	-356970
LCACR	318269
Beach time to unload (min)	-304388
LAV - 25	284617
TEU - shelter	-264102
Mean time to repair (min)	226123
Use Crane?	-201325
Use Austere Port?	-184289
Number of landing spots	180638
TEU - Eng Co	170005
EFSS Element	-167936
ACV speed (kts)	-137147
Time to load -crane (min)	-88045
Use Stern Ramp?	-82669
Time to load - stern (min)	58659
Time to load - port (min)	51559
CEB Element	-43346
Time to unload (min)	-15755
Transition time (min)	12135

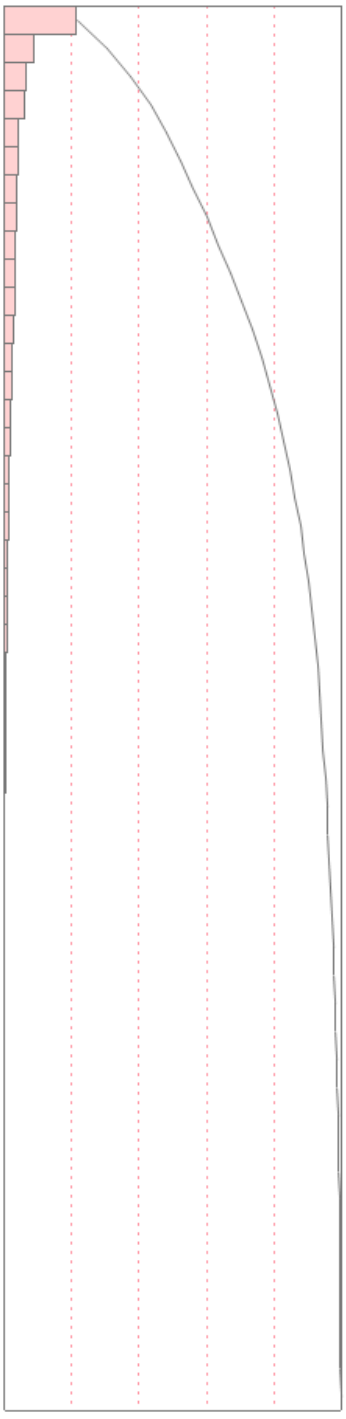


Figure 91: Pareto Plot for Fuel - Feasible Space, All Variables

15.5.2 Robust Design Conclusions

The focus of this chapter was to present the results of using the DELAS model in a robust design framework and compare those results across four sampling methods. The results of the robust design process were presented for a single scenario, a handful of scenarios, complete coverage of the noise space, and feasible scenario coverage. Although many of the design drivers remained the same throughout when considering performance, the loss drivers changed considerable. The magnitude of the variations drove the loss function instead of the performance and cost metrics. In future work, the calculation of the loss function could be modified to gain more insight. As a demonstration, this chapter showed the importance of considering the feasible noise space instead of the complete space.

In reality, there are many considerations that could be included in the robust design and this chapter only presents one example loss function. The cost of the ship itself would provide further insight into the trade-off between number available and maximum lift and area. The fuel usage cost should be further refined to include fuel usage based on speed and size of the ship. The fuel usages were estimates and need expert refinement as well as the possibility of making the fuel usage dependent on the amount of cargo loaded. The DELAS model does not include any consideration of exposure or possibility of the MEC being damaged or sunk while traveling. The robust analysis presented here is to serve as an example process which provides insight and could be expanded to incorporate other responses of interest.

The previous three chapters have worked to address the following experiments:

1. Identify measure of performance (MoPs) that are relevant to range of operations
2. Identify sampling methods for segmented spaces
3. Compare the feasibility of the sampling methods to complete coverage
4. Compare robustness results for complete coverage and feasible scenario options

5. Identify the design drivers for the MEC

Chapter 13 developed shortfall as a cross-scenario MoP paired with fuel usage as a cost metric. The challenge was to develop MoPs that were meaningful across the range of military operations. Chapter 14 compared four sampling methods, including the concept of feasible space, which samples a segmented area within the overall noise space. These sampling methods were executed and the robust results compared. Across all the sampling options, the number of conceptual connectors, as well as the speed, lift, and area capabilities of the MEC drove the performance. When the focus is on longer operations, the interface options are important, but with the inclusion of smaller operations, the total trip time is the driver. Selecting the sampling method for the noise space is key in determining the design drivers.

CHAPTER XVI

CONCLUSIONS

16.1 Summary of Work and Contributions

The objective of this work is a process that can quantitatively assess the impacts of new capabilities and vessels at the systems-of-systems level. This process addresses the need for the design of future naval platforms to account for the interoperability of a variety of heterogeneous systems and their role in a larger system-of-systems context. This new methodology must be able to handle diverse, disruptive technologies acting on multiple elements within the system-of-systems architecture. It must also be capable of capturing the complex interactions between elements or the architecture and must be able to assess the impacts of new systems such as the Medium Exploratory Connector (MEC). The method address the following gaps in existing methods:

- Breakdown of modeling problem into component
- Parametric scenarios
- Heterogeneous, interacting fleet
- Dynamic loading
- Dynamic routing
- Analyzing design requirements across multiple scenarios

These gaps lead to the following assertions and hypotheses through the consideration of the research questions developed in Chapter 3:

Assertions:

1. Interface selection, loading, and routing sub-problems are abstractable

2. Scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables

Hypotheses:

1. Introducing matrix formulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduce the effort required to integrate new assets

2. Knapsack loading is an efficient and robust approach for solving the loading sub-problem

3. Matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing

4. Feasible scenario robust analysis identifies the design drivers for a range of scenarios

The following sections summarize the results of investigating these assertions and hypotheses. The research questions are revisited to show how the experiment results have been able to answer the research questions. The observations, research questions, hypotheses, experiments, and conclusions are summarized in Table 33.

16.1.1 Sub-Problem Abstraction

The challenges of the Sea Basing problem could not be addressed by an existing vehicle routing or simulation method. There are portion of the problem that have been solved using existing methods and these techniques were brought together to address the overall, total problem. The following research questions described the gap:

Table 33: Summary of Key Thesis Points

Observation	Research Questions	Hypothesis / Assertion	Experiments	Conclusions
The complexity demands decomposition of the task into a set of more manageable design problems	Which sub-problems are abstractable and can be dealt with as separate problems?	Interface selection, loading, and routing sub-problems are abstractable	Experiments will be posed for each sub-problem	Loading and routing and abstracted
Model must capture multiple scenarios	Minimum amount of information to define a scenario? Can this information be defined by a set of input variables?	Scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables	Identify and quantify scenarios relevant to MEC design Develop generic cargo vector including minimally transportable units	Four example scenarios developed: major combat operation, small military operation, humanitarian mission, and sustainment operation, Generic cargo vector developed
A system-of-systems focus must consider interfaces between assets and fleet mix, Treating vessels as cargo objects will require new methods in data handling and modeling	How can the interfaces between assets be treated as design variables? How can changing the fleet mix not change structure of the model? Can cargo nodes be treated as objects - ability to track cargo and interface use?	Introducing matrix formulation into Discrete Event Simulations will enable the abstraction of sub-processes at an object level and reduce the effort required to integrate new assets	Runtime penalty for adding additional assets and cargo nodes, Validate and verify model results	Matrix formulation allows abstraction of sub-process and reduces level of effort to introduce new assets with minimal runtime penalty, Model results compared and explained

Observation	Research Questions	Hypothesis / Assertion	Experiments	Conclusions
Dynamic cargo loading enables parametric scenarios	Can mathematical optimization techniques be used for local level decisions?	Knapsack loading is an efficient and robust approach for solving the loading sub-problem	Compare algorithms to minimizes introduction of additional wait time Document ability to match cargo desired and cargo delivered	Hypothesis refined - knapsack and assignment formulation is efficient and does not introduce additional wait time
Routing is needed so the supply chain is a product of the assets present	Can routing decision be made using parameters tracked in the model? Can the logistics chain formulation be a byproduct of the selected mix of assets?	Matrix based predictive queuing and cargo algorithms can accurately predict queue times for dynamic routing	Level of effort to incorporate a change, Compare accuracy of queue prediction, selection of cargo object and interface for different algorithms	Four algorithms compared against baseline of pre-set routing, Predictive routing with reconsideration upon arrival at destination selected, Multiple unload points as traveling salesman added to dynamic loading
An approach is needed that facilitates design in a cross scenario, system-of-systems framework	What design parameters of a new connector are key to improving SoS performance? Common set of parameters across scenarios? Continuous or discrete scenarios? How can these parameters be identified when scenarios are uncertain?	Feasible scenario robust analysis identifies the design drivers for a range of scenarios	Identify measures of performance, Identify sampling of segmented spaces, Compare the feasibility of the sampling methods to complete coverage, Compare robustness results for complete coverage, and feasible scenario options, Identify the design drivers for the MEC	Shortfall and fuel usage as MoPs, Nearly orthogonal latin hypercube sampling, Sensitivities compared for single scenario, four scenarios, full coverage, and feasible space, Results different, noise sampling impacts key parameters, Noise parameters are large drivers

1. Different types of vehicles will complete different processes but are there common elements that can be abstracted?

2. Which sub-problems are abstractable and can be dealt with as separate problems?

3. Can the sub-processes discussed be treated as individual problems?

4. Will abstracting sub-processes decrease the effort required to integrate a new asset?

This thesis demonstrated the incorporation of the matrix formulation allowed the common process elements to be abstracted. The loading and routing sub-problem are abstracted and have separate research questions and hypotheses. The abstraction does reduce the effort required to integrate new assets, as was described in Section 7.3.

16.1.2 Scenario Definition

The need for parametric scenarios led to the following research questions:

1. What is the minimum amount of information needed to define a scenario?

2. Can this information be defined by a set of variables allowing a single model for several scenarios?

It hypothesis that scenarios can be fully defined by a scheduled set of demands, distances between locations, and physical characteristics that can be treated as input variables was demonstrated. Scenarios within this framework are defined in terms of:

1. Operation objective

2. Performance period

3. Participating units

4. Geographical layout - real or hypothetical

5. Performance plan - including required resources

These characteristics form the inputs to the DELAS model. The operation objective and required resources form the demand vector, quantified using the generic cargo vector. The generic cargo vector defines minimally transportable units that serve as the basic unit in the dynamic loading. The performance period is the length of the model run. Participating units are defined as inputs including the number and characteristics, such as speed and lift capabilities. The geographical layout of the scenario determines the distances, including Sea Base stand-off distance and distance to supply bases. Additional geographical information includes the potential for an intermediary port and the number and grouping of beach landing zones. By building the model using the scenario variables as inputs, instead of part of the model formulation, DELAS is able to capture and model a variety of operational scenarios. Generic cargo vector defines minimally transportable units that serve as the basic unit in the dynamic loading. The performance period is the length of the model run. Participating units are defined as inputs including the number and characteristics, such as speed and lift capabilities. The geographical layout of the scenario determines the distances, including Sea Base stand-off distance and distance to supply bases. Additional geographical information includes the potential for an intermediary port and the number and grouping of beach landing zones. By building the model using the scenario variables as inputs, instead of part of the model formulation, DELAS is able to capture and model a variety of operational scenarios.

16.1.3 Matrix Formulation

The gaps in current capabilities highlighted the need for a heterogeneous, interacting fleet highlights the need to model vessel interfaces as well as fleet mix in addition to traditional asset performance. The need to treat cargo vessels as objects with cargo

properties require a methods in data handling. These operations led to the following questions:

1. How can the interfaces between assets be treated as design variables and not a pre-set option?

2. How can changing the fleet mix not involve changing the structure of the model?

3. Can cargo nodes be treated as objects including the ability to track cargo and interface use?

Introducing matrix manipulation into Discrete Event Simulations enables the abstraction of sub-processes at an object level and reduces the effort required to integrate new assets. Matrix based decision making extracts the selection of loading and unloading interfaces from the vessel processes. Abstracting the interface selection also enable the modification of the interfaces available for a vessel to use without modification of the vessel process. Although increasing the run time slightly, the matrix based formulation greatly reduces the effort to introduce new assets. This formulation also allows the modification of decision making processes without the modifying each individual vessel process, which in turn enables the dynamic loading and routing algorithms. Matrix formulation incorporated into discrete event simulation creates a flexible and expandable framework.

16.1.4 Dynamic Loading

The need for parametric scenarios highlighted the need for dynamic loading as a single type of cargo or pre-set loadings can not capture the variety of scenarios of interest. This problem was identified as abstractable and led to the following question:

1. Can mathematical optimization techniques be used for local level decisions?

The use of matrices solves the handling of cargo assets as individual assets, leaving the subproblem of selecting the cargo to load on an individual connector. The

experimentation of loading algorithms was based on matching the cargo demand and not introducing additional wait time. A knapsack formulation alone did not match the cargo delivered to the cargo demanded. Combining an assignment problem with a knapsack loading formulation yields an algorithm that can overcome stalling of cargo selection and match the delivery of cargo. The hypothesis was disproven, but was demonstrated when modified to include an assignment formulation: A combined knapsack loading and assignment problem is an efficient and robust approach for solving the loading sub-problem. Realistic loading of vessels and other assets capture the cargo delivery capability of the modeled operation. Dynamic loading allows the scenario to be varied as well as the lift capabilities of the connectors.

16.1.5 Dynamic Routing

Dynamic routing is needed to allow for a reconfigurable supply chain to maintain a robust and flexible operation. This need led to the following questions:

1. Can routing decision be made using parameters tracked in the model?

2. Can the logistics chain formulation be a byproduct of the selected mix of assets?

The dynamic routing scheme can handle the loss or gain or a supplier or connectors and be able to choose the connector or to choose which route the connector should travel. This allows the model to not have a preset logistics flow but be represented as a network of possible nodes and connections. Four dynamic routing algorithms, based on concepts from traditional vehicle routing and computer science, were compared to a baseline of set routes. Routing based on predicted queue time but including the reconsideration of available interfaces upon arrival at the loading and unloading locations demonstrated its ability to route across a variety of test cases. The inclusion of a distribution on the wait times identified the strength of including reconsideration by taking advantage of changes that can speed the loading process. One probabilistic variable was tested and additional probabilistics, such as repairs, would further

improve the performance of the predictive with reconsideration algorithm. This algorithm is scalable to a heterogeneous mix of vessels of any size and number of supply nodes.

The possibilities of unloading at multiple beach groups is considered a special case. A traveling salesman type problem is added to the dynamic loading algorithm, so in addition to selecting the loading vessel, the unloading beach groups are selected as well as the total cargo to load on the connector. If hopping is available for a connector, the cargo selection algorithm will be replaced, adding to the total travel time in addition to outputting the cargo to load. The total travel time will be based on the predicted queue to the first beach location and the total trip time to complete the hops and unload times at selected locations. The option to visit multiple unload locations incorporated a traveling salesman problem into the dynamic loading algorithm and can be incorporated without changing the process for the dynamic routing algorithm.

16.1.6 Modeling Contribution

Incorporating the techniques investigated, results in the DELAS model and addresses the first five gaps highlighted from existing modeling techniques. The incorporation of matrix formulation into DES gives a more flexible way to handle large scale models. Matrices are an effective was of tracking properties across a heterogeneous fleet without making assumptions about the characteristics of the vessels. The global consideration of these properties allows for the abstraction of subproblems. The matrix formulation would be applicable for any large, heterogeneous system that requires decisions to be made during the object processes.

While more specific to the problem of focus for this thesis, the testing of dynamic loading and routing algorithms provides insight into the type of information and level of detail required to make decisions. The dynamic loading algorithm demonstrates the ability to incorporate traditional optimization techniques as a decision making

formulation within a DES. The testing of dynamic routing algorithms illustrated the capability to include algorithms from different fields of study into the decision making formulation. The testing and incorporation of these decision making algorithms shows that DES decision making is not limited to if-then statements but can be expanded to include techniques borrowed from other areas of expertise.

16.1.7 Robust Design

The need for an approach to facilitates design in a cross scenario, system-of-systems framework led to the following questions:

1. What design parameters of a new connector are key to improving the overall performance of the heterogeneous system?

2. Does a common set of parameters exist across multiple scenarios?

3. Should scenarios be defined using continuous variables or as a discrete selection? Is this a necessary model design choice?

4. How can these parameters be identified when scenarios are uncertain?

Section 12.5 highlighted the difficulty of identifying important trade-offs in a one-on-one off structure. The number of design parameters is great and inter-dependent so all combinations can not be explored in this manor. Section 15.1 and 15.2 demonstrated that the design drivers change with the scenarios of interest.

Exercising the DELAS model through the process of robust design fulfills the need of analyzing design requirements across multiple scenarios. To capture the impact of incorporating multiple scenarios, four experiments plans were carried through the robust design process. The first was based only on vessel design parameters for the conceptual new connector, analyzed for a singe scenario. This was compared to complete coverage of the noise space, which included scenario variables such as the number of other vessels present and the demanded cargo. A set of scenarios were

also considered as well as the concept of feasible noise space, where the noise space sampled was based on the scenarios of interest.

The performance metric of shortfall had number available, maximum lift, maximum area, and SES speed as design drivers. There was a trade-off in the number available and size or speed. When looking at the feasible space, the relationship between size and number available was strong enough to reverse the number available, to desiring fewer ships, if the ship was large enough. These design drivers remained constant across the sampling schemes but the secondary drivers changed. The change in factors was even greater when considering loss, a combination of the shortfall and fuel usage. In the full coverage and feasible space test, the variance in the responses ended up dominating the loss function. This lead to the exploration of the combined space, which highlighted the importance of the cargo units to deliver and the other vessels and vehicles present. The combined analysis provided the most obvious difference between the full and feasible coverage, highlighting the importance of sampling feasible regions.

16.1.8 Design Recommendations

Exploring the results of the robust design process, some design recommendations can be made for the MEC. It is important to note that these recommendation are based on the assumption developed throughout this thesis, especially the selection of the vessels and vehicles present for different scenarios, and are only based on considering shortfall and fuel usage. For the MCO operation, many, fast MECs are recommended with a maximum lift of at least 700 LT and max cargo deck area of 10000 sqft. It is important that the MEC has interface options and is reliable, not requiring maintenance more than once a day. When looking across the four scenarios, the lift and area capabilities are more importance than the number. There is a diminishing return on incorporating more MECs, especially over 7. The speed of interface is now

more important than options, so a single quick interface design that can quickly load at the Sea Base and unload at the beach would be be a better design choice. The full coverage of the design space indicated that more MECs introduce more variation in the performance, but having more available is important to the performance. The MEC needs to be able to carry at least 650 LT and 9000 sqft of cargo. For the wide variety of scenarios sampled here, it is important that an austere port can be used if one is present. The sensitivity to all variables indicated that the MEC needs to be able to climb a high enough beach slope to have multiple loading spots. Looking at the feasible space, the speed becomes more important than the number available, as long as the MEC can carry 750 LT and more than 9000 sqft. Again, it is more important to be able to quickly load and unload cargo rather than have loading options, but multiple unloading spots is important.

Looking across all of the results, common trends indicate that the design requirement is important independent of the sampling method selected. Decision makers can use the resultant trends to set design requirements for a robust MEC design and make acquisition decisions. Based on the work of this thesis, the MEC must be able to carry 700 LT and more than 9000 sqft of cargo. The recommendation toward a large MEC aligns with the candidate design explored in Section 12.5. The MEC should be able to travel quickly, but must also be able to travel with the fleet for long distance deployments. Unless the MEC will only be involved in MCO type scenarios, the improvement of the single type of interface is more important than having multiple interface options. The interface should be designed so the total time to interface, load, and separate is less than 3 hours. The time to beach, unload, and get off the beach should be minimized. Combining this need for quick beaching and the importance of the number of beach spots highlights the importance of the MECs climb ability. The unload challenge can be assisted with the option to unload at an austere port. The MEC must be reliable enough to be self-deployed for long distances and be able to

make multiple shorter trips between repairs. Overall, the MEC is recommended to be a larger ship that can quickly load and unload.

16.2 Future Work

The techniques included in the DELAS model have made considerable progress in the modeling of large scale logistics operations from a Sea Base. The user can capture a variety of scenarios and considerable an interacting, heterogeneous fleet. But, there is always improvements that could be made in modeling operations as they would be conducted by actual decision makers. The dynamic routing algorithm could be improved by including the possibility of reneging missions. This would include not dispatching a mission from a loading point or aborting the mission upon arrival at an unloading point. Reneging captures the possibility that a landing group or port is no longer available or no longer requires the cargo on board the connector. The dynamic loading algorithm could be expanded to include cubic considerations in addition to footprint and weight constraints. The cargo selection algorithm could be expanded so the loading times are dependent on the cargo to load in addition to the type of connector and interface selected.

The DELAS model only considers the movement of cargo toward the shore and ignores the return of cargo from the beach groups. This returned cargo could include injured personnel returning to a treatment ship or the return of broken or un-needed equipment for repair or storage. The returning cargo may not be delivered to the same ship or location as the next cargo loading. The ability to only visit a single asset at the Sea Base or staging bases exposes another weakness, the requirement that cargo must be loaded from a single location. In reality, connectors could visit multiple loading locations. Another interesting consideration would be the ability to transfer cargo between Sea Base elements.

There are many other factors that could be of interest and are not included in

the DELAS model. Currently repair is the only operational break for connectors, but manning considerations could be included for operational windows. It is assumed that repair can be done is a specific time frame, but in reality repair depends on the availability of spare parts. The tracking of spare parts available at location within the model could be tracked and incorporated into the routing decision algorithm.

As was seen in Section 12.5, there are operational decision not included in the analysis in the thesis. For example, the MEC does not have to travel with the rest of the fleet. This is an operation decision that may need to be investigated - is performance gained with the MEC traveling ahead? Are there security considerations? The fuel usage is currently a rough estimate but could be paired to ship design tools for basic sizing, fuel storage constraints, and costing. The MEC is a surface effect ship which has complex fuel and design considerations [104].

The DELAS model constructs a framework that is flexible and extensible. The next steps are possible within the existing framework by modifying existing decisions processes or adding new decision steps. That these decision would not change the overall framework of the model demonstrates the strengths of this formulation technique. The analysis completed in this thesis represents one use of the DELAS model to identify design drives for a potential new connector. The applications for this model are varied and range from investigating the fleet mix needed to complete an operation to identifying scenarios that challenge current capabilities.

16.3 Recommendations and Lessons Learned

This thesis presents a framework from improving operational level simulation and analysis. The key is establishing a framework that allows sub-problems to be abstracted and solved using mathematical and simulation techniques suited to those individual problems. Testing can be done at the sub-problem level to determine the best solution, but consideration must be given to the overall impact. While the

framework was applied to a Sea Base example, a similar decomposition can be applied to other large scale problems. Matrix formulation can be implemented when large amount of data must be maintained and used in decision making processes. The dynamic loading algorithm demonstrated how existing techniques for smaller scale problems can be used as decision algorithms in a larger framework. Local and predicted information is easily maintained in the matrices for decision support, i.e. calculations in the dynamic routing algorithm.

The concept of feasible robust design, where infeasible region of the design space are not included in the robust design process, can be applied to any scenario based problem. Cross scenario analysis will play an important role in future designs where a system must be able to perform in multiple scenarios. It is no longer possible to design a specific system for each scenario and a design that performs well across scenarios will be more highly valued than one that performs the best in one scenario.

Overall, the methods and experiments explored in this thesis represent a process for developing large scale modeling and exercising them for robust design across scenarios. The goal of the model must be determined as well as the scenarios of interest. The key scenario variables and design parameters are identified to determine the level or detail. The identification of decisions within the operation determine the abstractable processes, which can now be approached with separate solution algorithms. The complete operation model can then be used to make design or operational decisions with cross-scenario considerations.

APPENDIX A

TRADITIONAL SIMPY FORMULATION

This appendix is the traditional if-then decision process and resources management model. The traditional formulation is contrasted with the matrix formulation in Chapter 7. The traditional formulation starts with a list of inputs, in this case connector characteristics. The next section are general calculation and activation of the vessel processes. The vessel processes represent the operational steps of the connectors. The simulation section defines limited resources, representing the interface options.

```
from SimPy.Simulation import *

# Input data ----------

maxTime = 3200 # minutes
ARRint = 1.0 # time between MEC departures

sbdist = 150.00 # nmi
stddist = 150.00 # nmi initial stand off distance
NumMEC = 6 # Num MEC
NumMLP = 0
NumLMSR = 3

# MEC
SESspeed = 30.00 # kts
```

trandist = 1.00 # nmi

trantime = 30.00 # minutes

ACVspeed = 40.00 # kts

landingspot = 2

unloadtime = 45.00 # minutes

fromtrantime = 30.00 # minutes

MEC interface with Sea Base

rearloadspot = NumLMSR

sideloadspot = NumLMSR

mlploadspot = 0 # set to zero if can not mlp reload

reartimereload = 120 # minutes

sidetimereload = 120 # minutes

mlptimereload = 1200 # minutes

MLP

MLPspeed = 20.00 # kts

MLPstdoff = 25.00 # nmi

MLPballastdown = 30.00 # minutes

offloadLCAC = 30.00 # minutes

onloadLCAC = 30.00 # minutes

MLPballastup = 30.00 # minutes

MLPconnSB = 10.00 # minutes

MLPdisconnSB = 10.00 # minutes

LCAC

LCACspeed = 50.00 # kts

LCACunload = 45 # minutes

```
LCAClandingspot = 3
LCACreload = 60.00 # minutes

# Constants
LCACperMLP = 4

# Calculations ————————————-
stdtotrantime=(stddist-trandist)/SESspeed*60+trandist/ACVspeed*60 # minutes
sbtotrantime=(sbdist-trandist)/SESspeed*60 # minutes
timeonshore = unloadtime# minutes
MECtosb = trantime + (sbdist-trandist)/SESspeed*60 +trandist/ACVspeed*60 #
minutes

stdtoshore = (stddist-MLPstdoff)/MLPspeed*60 # minutes
sbtoshoretime = (sbdist-MLPstdoff)/MLPspeed*60 # minutes
shoretosb = (sbdist-MLPstdoff)/MLPspeed*60 # minutes

LCACtimetoshore= MLPstdoff/LCACspeed*60 # minutes

# Model components ————————————
class Source(Process):
    def generate(self,number, number2,TBA,resource, resource1, resource2, resource3,
resource4, resource5):
        for i in range(number):
            c = MEC(name = "'MEC%02d"'%(i,),sim=self.sim)
            self.sim.activate(c,c.MEC(shorespot=resource, rearspot=resource1, sidespot
= resource2, SBspot = resource3, MLPspot=resource5))
```

```
            yield hold,self,TBA
        for i in range(number2):
            c = MLP(name = "M%02d"%(i,),sim=self.sim)
            self.sim.activate(c,c.MLP(LCACshorespot=resource4, sidespot = resource2,
SBspot = resource3, MLPspot=resource5))
            yield hold,self,TBA


class MEC(Process):
    def MEC(self, shorespot, rearspot, sidespot, SBspot, MLPspot):
            yield hold,self,stdtotrantime
            while True:
                yield hold,self,trantime
                arrive = self.sim.now()
                yield request,self,shorespot
                wait = self.sim.now()-arrive
                MEC.wait +=wait
                yield hold,self,timeonshore
                yield release,self,shorespot
                MEC.numunloaded += 1
                yield hold,self,fromtrantime
                yield hold,self,MECtosb
                arrive1 = self.sim.now()
                yield request, self, SBspot,1

                rrear=min(rearspot.n,1.00/reartimereload)
                rside=min(sidespot.n, 1.00/sidetimereload)
```

```
                rmlp=min(mlploadspot,MLPspot.n,1.00/mlptimereload)
                rmax=max(rside,rrear,rmlp)

                if rmax==rrear:
                    yield request,self,rearspot
                    wait1 = self.sim.now()-arrive1
                    yield hold,self,reartimereload
                    yield release,self,rearspot
                elif rmax==rside:
                    yield request,self,sidespot,1
                    wait2 = self.sim.now()-arrive1
                    yield hold,self,sidetimereload
                    yield release,self,sidespot
                elif rmax==rmlp:
                    yield request,self,MLPspot,1
                    wait2 = self.sim.now()-arrive1
                    yield hold,self,mlptimereload
                    yield release,self,MLPspot
                else:
                    print "no spot selected - problem"
                yield release, self,SBspot
                yield hold,self,sbtotrantime


class MLP(Process):
    def MLP(self,LCACshorespot, sidespot, SBspot,MLPspot):
            yield request,self,MLPspot
            if stddist>MLPstdoff:
```

```
        yield hold,self,stdtoshore
        yield hold,self,MLPballastdown
        yield hold,self,offloadLCAC
    else:
        yield hold,self,0
    while True: # Generate LCACs
        for i in range(LCACperMLP):
            c = LCAC(name = "L%02d"%(i,),sim=self.sim)
            self.sim.activate(c,c.LCAC(LCACshorespot=LCACshorespot))

        yield get,self,self.sim.waitingLCAC,LCACperMLP
        yield hold,self,onloadLCAC
        yield hold,self,MLPballastdown
        yield hold,self,shoretosb
        arrive1 = self.sim.now()
        yield request, self, sidespot,2
        yield request, self, SBspot,2
        wait2 = self.sim.now()-arrive1
        MLP.wait+=wait2
        yield hold,self,MLPconnSB
        yield release,self,MLPspot
        yield release,self,SBspot
        # # IF MEC is waiting, load it first
        yield request, self,MLPspot,2
        yield request, self,SBspot,2
        yield hold,self,LCACreload
        yield hold,self,MLPdisconnSB
```

```
            yield release,self,sidespot
            yield release,self,SBspot

            if stddist>MLPstdoff:
                yield hold,self,stdtoshore
                yield hold,self,MLPballastdown
                yield hold,self,offloadLCAC
            else:
                yield hold,self,0

class LCAC(Process):
    def LCAC(self, LCACshorespot):
            yield hold,self,LCACtimetoshore
            arrive = self.sim.now()
            yield request,self,LCACshorespot
            wait3 = self.sim.now()-arrive
            MLP.wait2+=wait3
            yield hold,self,LCACunload
            yield release,self,LCACshorespot
            MLP.numunloaded += 1
            yield hold,self,LCACtimetoshore
            yield put,self,self.sim.waitingLCAC,[self]

# Model ----------------------
class SeaBaseModel(Simulation):
def run(self):
        self.initialize()
```

self.k = Resource(capacity=landingspot, name="Shore", unitName="Spot", sim=self)

self.rear = Resource(capacity=rearloadspot, name="Rear", unitName="Spot", sim=self, monitored=True)

self.side = Resource(capacity=sideloadspot, name="Side", unitName="Spot", sim=self, monitored=True)

self.SBspot = Resource(capacity=sideloadspot+rearloadspot, name="Side", unitName="Spot", sim=self, monitored=True)

self.LCACspot = Resource(capacity=LCAClandingspot, name="Side", unitName="Spot", sim=self, monitored=True)

self.MLPspot = Resource(capacity=NumMLP, name="Side", unitName="Spot", sim=self, monitored=True)

s = Source(sim=self)

waiting=[]

self.waitingLCAC=Store(capacity=100,initialBuffered=waiting,sim=self)

self.activate(s,s.generate(number=NumMEC, number2=NumMLP, TBA=ARRint, resource=self.k, resource1=self.rear, resource2=self.side, resource3=self.SBspot, resource4=self.LCACspot, resource5=self.MLPspot),at=0.0)

self.simulate(until=maxTime)

Experiment ——————————

MEC.numunloaded = 0

MEC.wait = 0

MEC.numatsb = 0

MLP.numunloaded = 0

MLP.wait = 0

MLP.wait2 = 0

MLP.numatsb = 0

MLP.MLParrive = 0

SeaBaseModel().run()

APPENDIX B

DELAS MODEL

The Discrete Event Logistics Advanced Simulation (DELAS) model represents the application of the concepts and techniques explored in Chapters 6 through 10. The inputs are read in a separate script as the inputs have been moved to comma delimited files. The input files include a listing of the vessel and vehicle attributes, such as speed and fuel burn. A distance file creates a python diction of distances between any two geographic location as well as the number of landing zones per beach group for each type of connector. The interface file lists the time requires to interface, load (or unload), and leave the interface option as well as the compatibilities between interface options. The final input file details the cargo, including the weights, areas, and priorities for each cargo item in the generic cargo vector. The cargo demand is broken down by demand period for each cargo item and each beach group. The initial load-outs of the cargo items, such as the cargo vessels and ISBs, are described in this file as well as the compatibility between the cargo items and the connectors. The input files were created so that a change, such as the addition of a cargo item, can be made in one file without modifying the model script.

Below is the main script for the DELAS model. Comments that describe the purpose of each process follow the definition of that process name. The discussion of these subroutines and vessel processes are discussed in Chapter 10.

```
    import sys
from SimPy.Simulation import *
import numpy as num
import csv as csv
```

```
from lpsolve55 import *
if sys.modules.haskey('inputreader'):
    reload(inputreader)
else:
    import inputreader
from inputreader import *

outputfile = open('results.csv','wb') #Writes to one file
w = csv.writer(outputfile)
doefile = open('doe.csv', 'r')
doereader = csv.reader(doefile)

try:
    headers = doereader.next()
    hasdoe = True
except:
    hasdoe = False

if hasdoe:
    scalarvar = []
    vectorvar = []
    vectorvarname = []
    vectorvarindex = []
    matrixvar = []
    matrixvarname = []
    matrixvarfirstindex = []
    matrixvarsecondindex = []
```

```
    for headerindex, header in enumerate(headers):
        numbracket = header.count('[')
        isscalar = (numbracket == 0)
        isvector = (numbracket == 1)
        ismatrix = (numbracket == 2)
        if numbracket > 2:
            print "[WARN] Can't handle a matrix with more than two dimensions"

        if isscalar:
            scalarvar.append(headerindex)
        elif isvector:
            vectorvar.append(headerindex)
            bracketindex = header.find('[')
            vectorvarname.append(header[:bracketindex])
            vectorvarindex.append(int(header[bracketindex + 1]))
        elif ismatrix:
            matrixvar.append(headerindex)
            bracketindex = header.find('[')
            matrixvarname.append(header[:bracketindex])
            matrixvarfirstindex.append(int(header[bracketindex + 1]))
            secondbracketindex = header.find('[', bracketindex + 1)
            matrixvarsecondindex.append(int(header[secondbracketindex + 1]))

def initglobalvars():
    global Names, Location, Goal, Predictedwaits2, Predictedcargo2, Spots2,
        Cargohave2, Cargowant2, Type2, Testloc, SBschedule,
```

```
        SBcargowant, PredictedSBschedule, Portcargowant,
        PredictedPortcargowant

    Names = []
    Location = []
    Goal = []
    Predictedwaits2 = []
    Predictedcargo2 = []
    Spots2 = []
    Cargohave2 = []
    Cargowant2 = []
    Type2 = []
    Testloc = []

    SBschedule = num.zeros(numcargo)
    SBcargowant = num.zeros(numcargo)
    PredictedSBschedule = num.zeros(numcargo)
    Portcargowant = num.zeros(numcargo)
    PredictedPortcargowant = num.zeros(numcargo)
    numships = 0
    for beach in beaches:
        globals()[beach+'CargoHave'] = num.zeros(numcargo)
        globals()['Predicted' + beach + 'schedule'] = num.zeros(numcargo)

hascase = True
while hascase:
    if hasdoe:
```

```
try:
    doecase = doereader.next()
except:
    hascase = False
    break
initglobalvars()

for var in scalarvar:
    # convert all variables start with 'num' to integer
    if headers[var].startswith('num'):
        vars()[headers[var]] = int(doecase[var])
    else:
        vars()[headers[var]] = float(doecase[var])

for var, name, index in zip(vectorvar, vectorvarname, vectorvarindex):
        vars()[name][index] = float(doecase[var])

for var, name, firstindex, secondindex in zip(matrixvar, matrixvarname, matrixvarfirstindex, matrixvarsecondindex):
    if name == "Distances":
        vars()[name][names[firstindex]][names[secondindex]] = float(doecase[var])
    elif name == "BeachTypes":
        TotalBeaches -= vars()[name][BeachTypeList[secondindex]][beaches[firstindex]]
        TotalBeaches += float(doecase[var])
        vars()[name][BeachTypeList[secondindex]][beaches[firstindex]] = float(doecase[var])
    else:
        vars()[name][firstindex][secondindex] = float(doecase[var])
```

```
else:
    hascase = False

checkvessel = SimEvent("Change Occured")

# Model components ------------------
class Source(Process):
    """Creates instances of Simpy Processes and activates them.

    Arguments are the respective number of each type of object in the simulation.
    """
    def generate(self, number1, number2, number3, number5,
                 number7, number8, number9, number12, number13,
                 number14, number15, number18, number19, number20,
                 number21, number22, number24, number25, number26, MLPloadcontrol):
        j = 0
        yield hold, self, 0
        c = writer(name = "write"%())
        activate(c, c.write1())
        c = writer2()
        activate(c, c.write2())
        c = CargoGenerator()
        activate(c, c.run())
        #Connectors - Supply Ships
```

```
        for i in range(number1):
            c = MEC("MEC%02d"%i, MECspots, MECwait,
                    MECmaxlift, MEClifteff, MECmaxarea,
                    MECareaeff, MECMTBF, MECMTTR,
                    MECcompatibility, MECSESspeed,
                    MECpushlocs, MECpulllocs,
                    [MECfuelidle, MECfuelSES, MECfuelACV], MECrange, MEChop-
ping)
            activate(c, c.run(j))
            Source.assets += 1
            j += 1
        for i in range(number7):
            c = JHSV("JHSV%02d"%i, JHSVspots, JHSVwait, JHSVmaxlift,
                    JHSVlifteff, JHSVmaxarea, JHSVareaeff, JHSVMTBF,
                    JHSVMTTR, JHSVcompatibility, JHSVspeed,
                    JHSVpushlocs, JHSVpulllocs,
                    [JHSVfuelidle, JHSVfuelusage], [], [])
            activate(c, c.run(j))
            Source.assets += 1
            j += 1
        for i in range(number21):
            c = TAKE("TAKE%02d"%i, TAKEspots, TAKEwait, TAKEmaxlift,
                    TAKElifteff, TAKEmaxarea, TAKEareaeff, TAKEMTBF,
                    TAKEMTTR, TAKEcompatibility, TAKEspeed,
                    TAKEpushlocs, TAKEpulllocs,
                    [TAKEfuelidle, TAKEfuelusage], [], [])
            activate(c, c.run(j))
```

```
        Source.assets += 1
        j += 1
    for i in range(number19):
        c = LSV("LSV%02d"%i, LSVspots, LSVwait, LSVmaxlift,
                LSVlifteff, LSVmaxarea, LSVareaeff, LSVMTBF,
                LSVMTTR, LSVcompatibility, LSVspeed, LSVpushlocs,
                LSVpulllocs, [LSVfuelidle, LSVfuelusage], [], [])
        activate(c, c.run(j))
        Source.assets += 1
        j += 1
    for i in range(number18):
        c = LCU2000("LCU2000%02d"%i, LCU2000spots, LCU2000wait,
                LCU2000maxlift, LCU2000lifteff, LCU2000maxarea,
                LCU2000areaeff, LCU2000MTBF, LCU2000MTTR,
                LCU2000compatibility, LCU2000speed,
                LCU2000pushlocs, LCU2000pulllocs,
                [LCU2000fuelidle, LCU2000fuelusage],[], [])
        activate(c, c.run(j))
        Source.assets += 1
        j += 1
    #Connectors - Aerial
    for i in range(number13):
        c = MV22("MV22%02d"%i, MV22spots, MV22wait, MV22maxlift,
                MV22lifteff, MV22maxarea, MV22areaeff, MV22MTBF,
                MV22MTTR, MV22compatibility, MV22speed,
                MV22pushlocs, MV22pulllocs,
                [MV22fuelidle, MV22fuelusage], MV22range, [])
```

```
        activate(c, c.run(j))
        Source.assets += 1
        j += 1
    #Connectors - Carried into theater
    for i in range(number22):
        c = INLS("INLS%02d"%i, INLSspots, INLSwait, INLSmaxlift,
                INLSlifteff, INLSmaxarea, INLSareaeff, INLSMTBF,
                INLSMTTR, INLScompatibility, INLSspeed,
                INLSpushlocs, INLSpulllocs,
                [INLSfuelidle, INLSfuelusage], INLSrange, [])
        activate(c, c.run(j, ISBINLS))
        Source.assets += 1
        j += 1
    for i in range(number26):
        c = LCU1600("LCU1600%02d"%i, LCU1600spots, LCU1600wait, LCU1600maxlift,
                LCU1600lifteff, LCU1600maxarea, LCU1600areaeff, LCU1600MTBF,
                LCU1600MTTR, LCU1600compatibility, LCU1600speed,
                LCU1600pushlocs, LCU1600pulllocs,
                [LCU1600fuelidle, LCU1600fuelusage], [], [])
        activate(c, c.run(j, ISBLCU1600))
        Source.assets += 1
        j += 1
    #Connectors - Helicopter
    for i in range(number14):
        c = CH46("CH46%02d"%i, CH46spots, CH46wait,
                CH46maxliftinternal, CH46maxliftsling,
                CH46lifteff, CH46maxareainternal,
```

```
                CH46maxareasling, CH46areaeff, CH46MTBF,
                CH46MTTR, CH46compatibility, [CH46cleanspeed, CH46slingspeed],
                CH46pushlocs, CH46pulllocs,
                [CH46fuelidle, CH46fuelclean, CH46fuelsling],
                CH46waitinternal, CH46waitsling, CH46range, [])
        activate(c, c.run(j, ISBCH46))
        Source.assets += 1
        j += 1
    for i in range(number15):
        c = CH53("CH53%02d"%i, CH53spots, CH53wait,
                CH53maxliftinternal, CH53maxliftsling,
                CH53lifteff, CH53maxareainternal,
                CH53maxareasling, CH53areaeff, CH53MTBF,
                CH53MTTR, CH53compatibility, [CH53cleanspeed, CH53slingspeed],
                CH53pushlocs, CH53pulllocs,
                [CH53fuelidle, CH53fuelclean, CH53fuelsling],
                CH53waitinternal, CH53waitsling, CH53range, [])
        activate(c, c.run(j, ISBCH53))
        Source.assets += 1
        j += 1
    #Connectors - Air Cushion Vehicle
    for i in range(number5):
        c = LCAC("LCAC%02d"%i, LCACspots, LCACwait, LCACmaxlift,
            LCAClifteff, LCACmaxarea, LCACareaeff, LCACMTBF,
            LCACMTTR, LCACcompatibility, LCACspeed, LCACpushlocs,
            LCACpulllocs, LCACrange,[LCACfuelidle, LCACfuelACV], [])
        activate(c, c.run(j, ISBLCAC, SBLCAC, beachemptyLCAC))
```

```
            Source.assets += 1
            j += 1
        for i in range(number9):
            c = LCACR("LCACR%02d"%i, LCACRspots, LCACRwait,
                    LCACRmaxlift, LCACRlifteff, LCACRmaxarea,
                    LCACRareaeff, LCACRMTBF, LCACRMTTR,
                    LCACRcompatibility, LCACRspeed, LCACRpushlocs,
                    LCACRpulllocs, LCACRrange, [LCACfuelidle, LCACfuelACV],
[])
            activate(c, c.run(j, ISBLCACR, SBLCACR, beachemptyLCACR))
            Source.assets += 1
            j += 1
        #Sea Base Cargo Ships
        for i in range(number3):
            c = LMSR("LMSR%02d"%i, LMSRspots, LMSRcargo, LMSRspeed,
                    [LMSRsmallhelospots, LMSRlargehelospots],
                    [LMSRsmallLCspots, LMSRlargeLCspots])
            activate(c, c.run(j))
            Source.assets += 1
            j += 1
        for i in range(number12):
            c = LHD("LHD%02d"%i, LHDspots, LHDcargo, LHDspeed,
                    [LHDsmallhelospots, LHDlargehelospots],
                    [LHDsmallLCspots, LHDlargeLCspots])
            activate(c, c.run(j))
            Source.assets += 1
            j += 1
```

```
for i in range(number24):
    c = LPD("LPD%02d"%i, LPDspots, LPDcargo, LPDspeed,
            [LPDsmallhelospots, LPDlargehelospots],
            [LPDsmallLCspots, LPDlargeLCspots])
    activate(c, c.run(j))
    Source.assets += 1
    j += 1
for i in range(number25):
    c = LSD("LSD%02d"%i, LSDspots, LSDcargo, LSDspeed,
            [LSDsmallhelospots, LSDlargehelospots],
            [LSDsmallLCspots, LSDlargeLCspots])
    activate(c, c.run(j))
    Source.assets += 1
    j += 1
#MLP
for i in range(number2):
    c = MLP("M%02d"%i)
    activate(c, c.MLP(j, MLPloadcontrol))
    Source.assets += 1
    j += 1
#Shore
for i in range(number8):
    c = ISB(name = "ISB%02d"%i)
    activate(c, c.run())
    Source.assets += 1
    j += 1
for i in range(number20):
```

```
            c = Port(name = "Port%02d"%i)
            activate(c, c.run(j))
            Source.assets += 1
            j += 1

        c = ShoreBeach([], [], [], [])
        activate(c, c.run())
        Source.assets += TotalBeaches
        j += TotalBeaches
        global numships
        numships= j

class writer(Process):
    """Calculations for results.

    write1 is the PEM.
    """
    def write1(self):
        item2 = []
        global TotalShortfall
        global CargoNeededOverall
        TotalShortfall = 0
        global Cargodelivered
        yield hold, self, Timebetween - 2
        while True:
            Shorecargohave = num.zeros(numcargo)
            for beach in beaches:
```

```
                cargohavetemp = globals()[beach+'CargoHave']
                Shorecargohave += cargohavetemp
            translate = num.multiply(cargomatrix, Shorecargohave)
            Cargo = num.sum(translate, axis = 1)
            Cargodelivered = num.sum(Cargo, axis = 0)
            Shortfall = num.maximum(CargoNeededOverall - Cargodelivered, 0)
            TotalShortfall += Shortfall
            Percentdelivered = Cargodelivered/CargoNeededTotal
            yield hold, self, Timebetween


    class writer2(Process):
        """Calculates and writes results to csv outfile.

        write2 is the PEM.
        """
        def init(self):
            Process.init(self)
            item = ('Time', 'MEC.numunloaded', 'MEC.travel', 'MEC.unloadqueue',
                    'MEC.loadqueue', 'MEC.load', 'LCAC.numunloaded', 'LCACR.numunloaded',
                    'CH46.numunloaded', 'CH53.numunloaded', 'Cargo[0]',
                    'Cargo[1]', 'Cargo[2]', 'Cargo[3]', 'Cargo[4]',
                    'Cargo[5]', 'Cargo[6]', 'Cargo[7]','Cargo[8]',
                    'Cargo[9]','Cargo[10]', 'CPItotal', 'num.average(MEC.loadwts)',
                    'num.average(MEC.loadareas)', 'num.average(LCAC.loadwts)','num.average(LCA
                    'num.average(LCACR.loadwts)', 'num.average(LCACR.loadareas)',
'num.average(CH46.loadwts)',
                    'num.average(CH46.loadareas)', 'num.average(CH53.loadwts)','num.average(CH53
```

```
                'deliverytime', 'TotalShortfall', 'Percentdelivered', 'Seafuel + Air-
fuel')
            #w.writerow(item)

        def write2(self):
            def average(array):
                if array:
                    return num.average(array)
                else:
                    return 0
            global deliverytime
            deliverytime = maxTime
            deliverycount = 0
            global Cargodelivered
            while True:
                yield hold, self, 60
                Shorecargohave = num.zeros(numcargo)
                for beach in beaches:
                    cargohavetemp = globals()[beach+'CargoHave']
                    Shorecargohave += cargohavetemp
                translate = num.multiply(cargomatrix, Shorecargohave)
                Cargo = num.sum(translate, axis = 1)
                Cargodelivered = num.sum(Cargo, axis = 0)
                CPItotal = num.sum(Shorecargohave * CPI)
                Percentdelivered = Cargodelivered / CargoNeededTotal
                if Percentdelivered > .99 and deliverycount == 0: #identify time when
all cargo has been delivered (overall, not day by day)
```

```
        deliverytime = now()
        deliverycount = 1

    MEC.fuel = sum([mec.fuel for mec in MEC.roster])
    MEC.numunloaded = sum([mec.numunloaded for mec in MEC.roster])
    MEC.travel = sum([mec.travel for mec in MEC.roster])
    MEC.unloadqueue = sum([mec.unloadqueue for mec in MEC.roster])
    MEC.loadqueue = sum([mec.loadqueue for mec in MEC.roster])
    MEC.load = sum([mec.load for mec in MEC.roster])
    for mec in MEC.roster:
        MEC.loadwts.extend(mec.loadwts)
        MEC.loadareas.extend(mec.loadareas)

    LCAC.fuel = sum([lcac.fuel for lcac in LCAC.roster])
    LCAC.numunloaded = sum([lcac.numunloaded for lcac in LCAC.roster])
    for lcac in LCAC.roster:
        LCAC.loadwts.extend(lcac.loadwts)
        LCAC.loadareas.extend(lcac.loadareas)

    LCACR.fuel = sum([lcacr.fuel for lcacr in LCACR.roster])
    LCACR.numunloaded = sum([lcacr.numunloaded for lcacr in LCACR.roster])
    for lcacr in LCACR.roster:
        LCACR.loadwts.extend(lcacr.loadwts)
        LCACR.loadareas.extend(lcacr.loadareas)

    CH46.fuel = sum([ch46.fuel for ch46 in CH46.roster])
    CH46.numunloaded = sum([ch46.numunloaded for ch46 in CH46.roster])
```

```
for ch46 in CH46.roster:
    CH46.loadwts.extend(ch46.loadwts)
    CH46.loadareas.extend(ch46.loadareas)

CH53.fuel = sum([ch53.fuel for ch53 in CH53.roster])
CH53.numunloaded = sum([ch53.numunloaded for ch53 in CH53.roster])
for ch53 in CH53.roster:
    CH53.loadwts.extend(ch53.loadwts)
    CH53.loadareas.extend(ch53.loadareas)

JHSV.fuel = sum([jhsv.fuel for jhsv in JHSV.roster])
LSV.fuel = sum([lsv.fuel for lsv in LSV.roster])
LCU2000.fuel = sum([lcu2000.fuel for lcu2000 in LCU2000.roster])
TAKE.fuel = sum([take.fuel for take in TAKE.roster])
INLS.fuel = sum([inls.fuel for inls in INLS.roster])
MV22.fuel = sum([mv22.fuel for mv22 in MV22.roster])
LCU1600.fuel = sum([lcu1600.fuel for lcu1600 in LCU1600.roster])

Seafuel = (MEC.fuel + LCAC.fuel + LCACR.fuel + MLP.fuel + JHSV.fuel
+
            LSV.fuel + LCU2000.fuel + TAKE.fuel + INLS.fuel + LCU1600.fuel)
Airfuel = MV22.fuel + CH46.fuel + CH53.fuel
item = (now(), MEC.numunloaded, MEC.travel, MEC.unloadqueue,
        MEC.loadqueue, MEC.load, LCAC.numunloaded, LCACR.numunloaded,
        CH46.numunloaded, CH53.numunloaded, Cargo[0], Cargo[1], Cargo[2],
        Cargo[3], Cargo[4], Cargo[5], Cargo[6], Cargo[7], Cargo[8],
        Cargo[9], Cargo[10], CPItotal, average(MEC.loadwts),
```

```
                average(MEC.loadareas), average(LCAC.loadwts), average(LCAC.loadareas),
                average(LCACR.loadwts), average(LCACR.loadareas), average(CH46.loadwts)
                average(CH46.loadareas), average(CH53.loadwts), average(CH53.loadareas),
                deliverytime, TotalShortfall, Percentdelivered, Seafuel + Airfuel)
            #w.writerow(item)
            if self.sim.now() == maxTime-1:
                w.writerow(item)

    #Properties of connectors
    class Organic:
        """Property to designate an organic connector.

        An organic connector is not self deployed and must be ferried into theater
by another vessel.
        Implemented by: CarriedToTheaterConnector and Helicopter.
        """
        pass

    class Aerial:
        """Property to designate an aerial connector.

        An aerial connector can only repair once landed.
        Implemented by: MV22.
        """
        pass

    class Surface:
```

"""Property to designate a surface connector.

A surface connector can repair as a self contained unit.

Implemented by: SupplyShip, MEC, CarriedToTheaterConnector.

"""

pass

class Connector(Process):

"""Generic process for a connector.

Connectors move cargo to meet a demand. The PEM is run.

These connectors are assumed to start equally distributed between the Intermediate Staging Bases (ISB) with no cargo on board.

"""

def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, vehiclerange, hopping):

Process.init(self)

self.name = name

self.fuel = 0

self.load = 0

self.trip = 0

self.travel = 0

self.numunloaded = 0

self.repaircount = 0

self.unloadqueue = 0

self.loadqueue = 0

```
self.cargohave = num.zeros(numcargo)
self.cargowant = num.zeros(numcargo)
self.pullwaittime = num.array([])
self.pushwaittime = num.array([])
self.predictedcargo = num.array([])
self.spots = spots
self.wait = wait
self.lift = maxlift*lifteff
self.area = maxarea*areaeff
self.MTBF = MTBF
self.MTTR = MTTR
self.compatibility = compatibility
self.speed = speed
self.fuelusage = fuelusage
self.pushlocs = pushlocs
self.pulllocs = pulllocs
self.range = vehiclerange
self.hopping = hopping
self.nexthop = []
self.nexthop2 = []
self.cargohavehop = num.zeros(numcargo)
self.arrivedSignal = SimEvent()
self.unloadedSignal = SimEvent()
self.SBSignal = SimEvent()
self.loadwts = []
self.loadareas =[]
```

```
def run(self, j, store=[]):
    """The PEM for connectors.

    Steps:
    1. A mission is selected, establishing locations to pick up and drop of cargo.
    2. The connector travels to the pick-up location and identifies a spot on a
ship, at a port, or at a supply base to load cargo.
    3. After traveling to the push goal, a spot is selected and the cargo of-
floaded. At this point, the connector identifies its
    next mission and travels to a cargo supply point to load again.
    4. Switch between pushing/pulling cargo and repeat steps.
    """
    namematrix(self)
    yield hold, self, 1

    if isinstance(self, Organic):
        yield put, self, store, [self]
        yield waitevent, self, checkvessel

    hasgoal = choosegoal(self)
    if self.hopping == 1 and len(self.nexthop)>0:
        self.definehop()
    while not hasgoal:
        yield waitevent, self, checkvessel
        hasgoal = choosegoal(self)
        if self.hopping == 1 and len(self.nexthop)>0:
            self.definehop()
```

```
while True:
    time1 = now()
    if isinstance(self, Surface):
        # If the connector needs repair at SB or ISB, wait until fixed
        if self.goal.strip("0123456789") in ["SB", "ISB"]:
            if self.repaircount >= self.MTBF:
                yield hold, self, self.MTTR
                self.repaircount = 0
    # choose cargo and find a compatible interface
    hascargoandinterface = False
    while not hascargoandinterface:
        if self.purpose == "pull":
            if self.hopping == 0:
                [schedule, priority] = idschedule(self, self.nextpush)
                self.cargowant, loadoutarea, loadoutwt, relativepriority = choose-
cargo(self, schedule, priority, self.goal)
            else:
                #self.nexthop2 = [self.nextpush, self.nexthop]
                self.cargowant, loadoutarea, loadoutwt, relativepriority, bea-
chorder, beachnames, totaltriptime = choosehops(self, self.goal, False)
                self.definehop2(beachnames)

        cargoatlocation = calcloc(self, self.goal)
        spotswithcargo = checkspotcargo(self, cargoatlocation)
        hascargoandinterface = num.amax(spotswithcargo)
        if not hascargoandinterface:
```

```
            yield waitevent, self, checkvessel

row, col, wait, incomp = findspot(self, spotswithcargo, j)

time2 = now() - time1
self.repaircount += time2
self.fuel += self.getspentfuel([time2], [self.fuelusage[0]])
if self.location.strip("0123456789") in self.pulllocs:
    self.loadqueue += time2
else:
    self.unloadqueue += time2

if self.purpose == "pull":
    self.loadareas.append(loadoutarea)
    self.loadwts.append(loadoutwt)
    updateschedule(self)

if isinstance(self, Aerial):
    # If the needs repair at SB or ISB, wait until fixed
    if self.goal.strip("0123456789") in ["SB", "ISB"]:
        if self.repaircount >= self.MTBF:
            yield hold, self, self.MTTR
            self.repaircount = 0

yield hold, self, wait
self.load += wait
self.trip += 1
```

```
                Spots[row, col] = 1
                Spots[row] = Spots[row] + incomp

                hasgoal = self.changemission(row, j)

                if self.location.strip("0123456789") in self.pushlocs or self.location in
self.pushlocs:
                    self.numunloaded += 1
                while hasgoal == 0:
                    yield waitevent, self, checkvessel
                    hasgoal = choosegoal(self)
                    if self.hopping == 1 and len(self.nexthop)>0:
                        self.definehop()

                checkvessel.signal()
                yield hold, self, 0

                traveltime = self.gettraveltime()
                totaltraveltime = sum(traveltime)

                yield hold, self, totaltraveltime

                self.repaircount += totaltraveltime
                self.travel += totaltraveltime
                self.fuel += self.getspentfuel(traveltime, self.fuelusage)

        def gettraveltime(self):
```

```
        """Calculate travel time."""
        dist = Distances[self.location][self.goal]
        traveltime = dist / self.speed * 60
        return [traveltime]

    def getspentfuel(self, time, usagerate):
        """Calculate fuel usage."""
        return sum([t*rate for t, rate in zip(time, usagerate)])

    def changemission(self, row, j):
        """Modify object properties for the next mission."""
        global Predictedwaits
        global Predictedcargo
        global PredictedSBschedule
        global PredictedPortcargowant

        if self.purpose == "pull":
            self.location = self.goal
            self.purpose = "push"
            if self.pullwaittime.size != 0:
                Predictedwaits = Predictedwaits - self.pullwaittime
            if self.predictedcargo.size != 0:
                Predictedcargo = Predictedcargo - self.predictedcargo
        else:
            if self.hopping == 1 and len(self.nexthop) >0:
                self.location = self.goal
                self.definehop()
```

```
                Cargowant[row] = Cargowant[row] - globals()['cargowant' + self.location
+ self.name]
                Cargohave[row] = Cargohave[row] + globals()['cargowant' + self.location
+ self.name]
                self.cargohavehop = self.cargohavehop - globals()['cargowant' + self.location
+ self.name]

            else:
                Cargowant[row] = Cargowant[row] - num.array(self.cargohave)
                Cargohave[row] = Cargohave[row] + num.array(self.cargohave)
                self.location = self.goal
                self.purpose = "pull"
                self.cargowant = num.zeros(numcargo)
                self.cargohave = num.zeros(numcargo)
                if self.pushwaittime.size != 0:
                    Predictedwaits = Predictedwaits - self.pushwaittime
            #remove cargo from the predicted schedule since the actual transaction
has occured
            if self.predictedcargo.size != 0:
                locstripped = self.location.strip("0123456789")
                if(locstripped == "SB"):
                    PredictedSBschedule -= num.sum(self.predictedcargo, 0)
                elif(locstripped == "Port"):
                    PredictedPortcargowant -= num.sum(self.predictedcargo, 0)
                else:
                    globals()['Predicted' + self.location +'schedule'] -= globals()['cargowant'
```

```
+ self.location + self.name]
            #choose new goal if next mission is not already decided
            if(self.purpose == "push"):
                self.goal = self.nextpush
                hasgoal = True
            else:
                hasgoal = choosegoal(self)
                if self.hopping == 1 and len(self.nexthop)>0:
                    self.definehop()

            replacematrix(self, j)

            return hasgoal

        def deploy(self, vehiclelist):
            """Initial assignment of vessels to start locations."""
            vehicleid = len(vehiclelist) - 1
            if vehicleid % numISB == 0:
                locationid = 0
            else:
                vehicle = vehiclelist[-2]
                previousid = vehicle.location[-2:]
                locationid = int(previousid) + 1
            self.location = "start%02d"%locationid

        def definehop(self):
            """Cycle through hop schedule"""
```

```
            self.nextpush = self.nexthop[0]
            self.nexthop.pop(0)
            self.cargohave = globals()['cargowant' + self.nextpush + self.name]

        def definehop2(self, beachnames):
            """Updated predicted values based on hop re-evaluation at load location"""
            cargowantcalc = self.cargowant
            for beach in beaches:
                globals()['Predicted' + beach +'schedule'] -= globals()['cargowant' + beach + self.name]
                globals()['cargowant' + beach + self.name] = num.zeros(numcargo)

            for beach in beachnames:
                globals()['cargowant' + beach + self.name] = num.minimum(cargowantcalc, globals()[beach + 'schedule'])
                cargowantcalc = cargowantcalc - globals()['cargowant' + beach + self.name]
                globals()['Predicted' + beach +'schedule'] += globals()['cargowant' + beach + self.name]

            self.nexthop = beachnames
            self.definehop()

    class SupplyShip(Surface, Connector):
        """Subclass of Connector and has Surface properties.
```

All of the supply ships are surface connectors. The MEC is the only class

that must change the general process.

Implemented by MEC, JHSV, TAKE, LSV, LCU-2000.

"""

def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, vehiclerange, hopping):

Connector.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, vehiclerange, hopping)

self.location = "start"

self.goal = "ISB00"

self.purpose = "pull"

self.type = "connector"

class MEC(SupplyShip):

"""Subclass of SupplyShip for the MEC connector.

Overrides the Connector gettraveltime method to account for change in operational mode.

"""

roster = []

def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, vehiclerange, hopping):

SupplyShip.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, vehiclerange, hopping)

```
            MEC.roster.append(self)
            self.deploy(MEC.roster)

        def gettraveltime(self):
            """Calculate travel time.

            Overrides the connector travel time routine since MEC has different travel
time."""
            dist = Distances[self.location][self.goal]
            transitiontime = MECtranstime
            traveltimeSES = (dist - MECtransdist) / MECSESspeed * 60
            traveltimeACV = MECtransdist / MECACVspeed * 60
            return [transitiontime, traveltimeSES, traveltimeACV]


    class JHSV(SupplyShip):
        """Subclass of SupplyShip for the JHSV."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                    MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
vehiclerange, hopping):
            SupplyShip.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                                MTBF, MTTR, compatibility, speed, pushlocs, pulllocs,
fuelusage,
                                vehiclerange, hopping)
            JHSV.roster.append(self)
            self.deploy(JHSV.roster)
```

```
    class TAKE(SupplyShip):
        """Subclass of SupplyShip for the TAKE."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                    MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
vehiclerange, hopping):
            SupplyShip.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                            MTBF, MTTR, compatibility, speed, pushlocs, pulllocs,
fuelusage,
                            vehiclerange, hopping)
            TAKE.roster.append(self)
            self.deploy(TAKE.roster)


    class LSV(SupplyShip):
        """Subclass of SupplyShip for the LSV."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                    MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
                    vehiclerange, hopping):
            SupplyShip.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                            MTBF, MTTR, compatibility, speed, pushlocs, pulllocs,
fuelusage,
                            vehiclerange, hopping)
            LSV.roster.append(self)
            self.deploy(LSV.roster)
```

```
    class LCU2000(SupplyShip):
        """Subclass of SupplyShip for the LCU2000."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
                 vehiclerange, hopping):
            SupplyShip.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                            MTBF, MTTR, compatibility, speed, pushlocs, pulllocs,
fuelusage,
                            vehiclerange, hopping)
            LCU2000.roster.append(self)
            self.deploy(LCU2000.roster)

    class MV22(Aerial, Connector):
        """Subclass of Connector with Aerial properties.

        The MV-22 is handled as an aerial connector, as it is assumed it can queue in
the air.
        If it is determined to act more like a helicopter, where landing is a priority,
not matching cargo,
        then the process can be switched to that of the helicopter class.
        """
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
vehiclerange, hopping):
            Connector.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
```

```
                MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fu-
elusage, vehiclerange, hopping)
        self.goal = "ISB00"
        self.purpose = "pull"
        self.type = "connector"
        MV22.roster.append(self)
        self.deploy(MV22.roster)


class CarriedToTheaterConnector(Organic, Surface, Connector):
    """Subclass of Connector with Organic and Surface properties.

    These connectors must be carried into theater on designated connectors.
    Once the ferrying ship arrives in theater, these connectors are offloaded first.
    These connectors then become independent vessels that travel between set
points.

    Implemented by INLS, LCU1600
    """
    def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
            MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
vehiclerange, hopping):
        Connector.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fu-
elusage, vehiclerange, hopping)
        self.location = "SBstart"
        self.goal = "SB"
        self.purpose = "pull"
```

```
        self.type = "connector"


    class INLS(CarriedToTheaterConnector):
        """Subclass of CarriedToTheaterConnector for the INLS."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
vehiclerange, hopping):
            CarriedToTheaterConnector.init(self, name, spots, wait, maxlift, lifteff,
maxarea, areaeff,
                                           MTBF, MTTR, compatibility, speed, pushlocs,
pulllocs, fuelusage, vehiclerange, hopping)
            INLS.roster.append(self)


    class LCU1600(CarriedToTheaterConnector):
        """Subclass of CarriedToTheaterConnector for the LCU1600."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage,
hopping):
            CarriedToTheaterConnector.init(self, name, spots, wait, maxlift, lifteff,
maxarea, areaeff,
                                           MTBF, MTTR, compatibility, speed, pushlocs,
pulllocs, fuelusage, vehiclerange, hopping)
            LCU1600.roster.append(self)


    class Helicopter(Organic, Connector):
```

"""Subclass of Connector with Organic properties.

Helicopters must be carried aboard another vessel to reach the Sea Base.

Once deployed, they follow a slightly modified procedure because they can not queue as an individual asset.

Priority is placed on landing not matching cargo.

Implemented by CH-46, CH-53.

"""

```
def init(self, name, spots, wait, maxliftinternal,
         maxliftsling, lifteff, maxareainternal,
         maxareasling, areaeff, MTBF, MTTR, compatibility,
         speed, pushlocs, pulllocs, fuelusage, waitinternal,
         waitsling, vehiclerange, hopping):
    Connector.init(self, name, spots, wait, 0, 0, 0, 0, MTBF, MTTR,
         compatibility, speed, pushlocs, pulllocs, [], vehiclerange, hopping)
    self.unloadqueue = 0
    self.loadqueue = 0
    self.location = "SBstart"
    self.goal = "SB"
    self.purpose = "pull"
    self.type = "connector"
    self.cargotype = ""
    self.liftinternal = maxliftinternal*lifteff
    self.liftsling = maxliftsling*lifteff
    self.areainternal = maxareainternal*areaeff
    self.areasling = maxareasling*areaeff
```

```
        self.lift = 0
        self.area = 0
        self.speedoptions = speed
        self.waitinternal = waitinternal
        self.waitsling = waitsling
        self.fuelidle = fuelusage[0]
        self.fuelclean = fuelusage[1]
        self.fuelsling = fuelusage[2]

    def run(self, j, store):
        """The PEM for Helicopters.  This method overrides the Connector run
method.

        If a landing spot can not be identified that has the cargo the helicopter
would like to carry or drop off,
        depending on the mission, the helicopter will land at the first available
spot.
        This is to minimize the time in the air.  Any repairs needed are completed
once the helicopter lands.
        Once the helicopter has landed, it will wait until the cargo becomes avail-
able at that source and will not
        change cargo supply locations to match its cargo needs.
        """
        namematrix(self)
        yield hold, self, 1
        yield put, self, store, [self]
        yield waitevent, self, self.SBSignal
```

```
            hasgoal = choosegoal(self)
            while not hasgoal:
                yield waitevent, self, checkvessel
                hasgoal = choosegoal(self)

            while True:
                time1 = now()
                # Since wait, row, col, and incomp are inside the if loop, without ini-
tialization
                # 'yield hold, self, wait + waitload' causes an error.
                wait = 0
                row = 0
                col = 0
                incomp = 0
                waitload = 0

                self.cargotype = ""
                if self.purpose == "pull":
                    [waitload, loadoutarea, loadoutwt, self.cargowant, relativepriority]
= self.choosehelocargo(False)
                cargoatlocation = calcloc(self, self.goal)
                spotswithcargo = checkspotcargo(self, cargoatlocation)
                hasspotwithcargo = num.amax(spotswithcargo)
                # Helo should land if available spot, not hover waiting for cargo
                if self.purpose == "pull" and not hasspotwithcargo:
                    spotexists = False
                    while not spotexists:
```

```
                self.cargowant = num.zeros(numcargo)
                cargoatlocation = calcloc(self, self.goal)
                spotswithcargo = checkspotcargo(self, cargoatlocation)
                spotexists = num.amax(spotswithcargo)

                if spotexists > 0 and hasspotwithcargo == 0:
                    row, col, wait, incomp = findspot(self, spotswithcargo, j)
                    checkcargo = 0

                    goalstripped = self.goal.strip("0123456789")
                    if goalstripped in ["SB", "ISB"]:
                        if self.repaircount >= self.MTBF:
                            yield hold, self, self.MTTR
                            self.repaircount = 0

                    while checkcargo == 0 and hasspotwithcargo == 0:
                        #check if can take internal load
                        self.lift = self.liftinternal
                        self.area = self.areainternal
                        [schedule, priority] = idschedule(self, self.nextpush)
                        self.cargowant, loadoutarea, loadoutwt, relativepriority =
choosecargo(self, schedule, priority, self.goal)
                        cargotest1 = Cargohave[row,:] - self.cargowant
                        if num.min(cargotest1) >= 0:
                            Cargohave[row,:] = Cargohave[row,:] - self.cargowant
                            Cargowant[row] = Cargowant[row] + num.array(self.cargowant)
                            self.cargohave = self.cargowant
```

```
                    self.cargowant = num.zeros(numcargo)
                    replacematrix(self, j)
                    self.cargotype = "Internal"
                    waitload = self.waitinternal
                    checkcargo = 1
                else:
                    #check if can take sling load
                    self.lift = self.liftsling
                    self.area = self.areasling
                    self.cargowant, loadoutarea, loadoutwt, relativeprior-
ity = choosecargo(self, schedule, priority, self.goal)
                    cargotest1 = Cargohave[row,:] - self.cargowant
                    if num.min(cargotest1) >= 0:
                        Cargohave[row,:] = Cargohave[row,:] - self.cargowant
                        Cargowant[row] = Cargowant[row] + num.array(self.cargowant)
                        self.cargohave = self.cargowant
                        self.cargowant = num.zeros(numcargo)
                        replacematrix(self, j)
                        self.cargotype = "Sling"
                        waitload = self.waitsling
                        checkcargo = 1
                if checkcargo == 0:
                    yield waitevent, self, checkvessel

            else:
                yield waitevent, self, checkvessel
                self.cargotype = ""
```

```
                    if self.purpose == "pull": #Recalculate cargo to load
                        [waitload, loadoutarea, loadoutwt, self.cargowant, relative
priority] = self.choosehelocargo(False)
                    cargoatlocation = calcloc(self,self.goal)
                    spotswithcargo = checkspotcargo(self,cargoatlocation)
                    hasspotwithcargo = num.amax(spotswithcargo)
                    if hasspotwithcargo > 0:
                        spotexists = 1
                        row, col, wait, incomp = findspot(self, spotswithcargo, j)
                time2 = now() - time1

            else:
                while hasspotwithcargo == 0:
                    yield waitevent, self, checkvessel
                    cargoatlocation = calcloc(self, self.goal)
                    spotswithcargo = checkspotcargo(self, cargoatlocation)
                    hasspotwithcargo = num.amax(spotswithcargo)
                row, col, wait, incomp = findspot(self, spotswithcargo, j)
                time2 = now() - time1

            if self.purpose == "pull":
                self.loadareas.append(loadoutarea)
                self.loadwts.append(loadoutwt)
                updateschedule(self)

            goalstripped = self.goal.strip("0123456789")
            if goalstripped in ["SB", "ISB"]:
```

```
                if self.repaircount >= self.MTBF:
                    yield hold, self, self.MTTR
                    self.repaircount = 0

                self.fuel += time2*self.fuelidle
                yield hold, self, wait + waitload
                self.load += wait + waitload

                if self.location.strip("0123456789") in self.pushlocs or self.location in
self.pushlocs:
                    self.numunloaded += 1
                    self.unloadqueue += time2
                    self.repaircount += time2
                    self.cargotype = ""
                else:
                    self.loadqueue += time2
                    self.repaircount += time2

                Spots[row, col] = 1
                Spots[row] = Spots[row] + incomp

                checkvessel.signal()
                yield hold, self, 0

                hasgoal = self.changemission(row, j)
                while not hasgoal:
                    yield waitevent, self, checkvessel
```

```
            hasgoal = choosegoal(self)

        if self.purpose == "pull": #travels to pull location in clean mode
            self.speed = self.speedoptions[0]
        dist = Distances[self.location][self.goal]
        traveltime1 = dist/self.speed*60
        yield hold, self, traveltime1

        if self.cargotype == "Sling":
            self.fuel += traveltime1*self.fuelsling
        else:
            self.fuel += traveltime1*self.fuelclean

        self.travel += traveltime1
        self.repaircount += traveltime1

def choosehelocargo(self, Predictive):
    """Choose cargo to load by checking internal loading then sling loading.

    Method changes self.lift, self.area, self.cargowant, self.cargotype.

    Returns:
    wait - loading time based on choice of internal/sling loading
    loadoutarea - area of cargo chosen
    loadoutwt - weight of cargo chosen
    """
    self.cargotype = "Internal"
```

```
            wait = self.waitinternal
            self.lift = self.liftinternal
            self.area = self.areainternal
            self.speed = self.speedoptions[0]
            if Predictive == True:
                schedule, priority = idpredictedschedule(self, self.nextpush)
            else:
                [schedule, priority] = idschedule(self, self.nextpush)
            self.cargowant, loadoutarea, loadoutwt, relativepriority = choosecargo(self,
schedule, priority, self.goal)

            if num.min(self.cargowant) > 100000:
                self.cargotype = "Sling"
                wait = self.waitsling
                self.lift = self.liftsling
                self.area = self.areasling
                self.speed = self.speedoptions[1]
                self.cargowant, loadoutarea, loadoutwt, relativepriority = choosecargo(self,
schedule, priority, self.goal)

            return [wait, loadoutarea, loadoutwt, self.cargowant, relativepriority]

    class CH46(Helicopter):
        """Subclass of Helicopter for the CH-46."""
        roster = []
        def init(self, name, spots, wait, maxliftinternal, maxliftsling,
                 lifteff, maxareainternal, maxareasling, areaeff, MTBF,
```

MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, waitinternal, waitsling, vehiclerange, hopping):

Helicopter.init(self, name, spots, wait, maxliftinternal, maxliftsling,

lifteff, maxareainternal, maxareasling, areaeff, MTBF,

MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, waitinternal, waitsling, vehiclerange, hopping)

CH46.roster.append(self)

class CH53(Helicopter):

"""Subclass of Helicopter for the CH-53."""

roster = []

def init(self, name, spots, wait, maxliftinternal, maxliftsling,

lifteff, maxareainternal, maxareasling, areaeff, MTBF,

MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, waitinternal, waitsling, vehiclerange, hopping):

Helicopter.init(self, name, spots, wait, maxliftinternal, maxliftsling,

lifteff, maxareainternal, maxareasling, areaeff, MTBF,

MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, waitinternal, waitsling, vehiclerange, hopping)

CH53.roster.append(self)

class AirCushionVehicle(Connector):

"""Subclass of Connector.

The main distinction from the general connector is the MLP have the option

to ferry the ACVs closer to shore if the SB Cargo Ships are outside their range.

Implemented by LCAC, LCAC-R

"""

def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, vehiclerange, fuelusage, range2, hopping):

Connector.init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,

MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, fuelusage, [], hopping)

self.loadqueue = 0

self.unloadqueue = 0

self.MLPqueueSB = 0

self.MLPqueuebeach = 0

self.location = "ISB00"

self.goal = "SB"

self.purpose = "pull"

self.type = "connector"

self.vehiclerange = vehiclerange #can be ferried so different from self.range usage for general connector

self.fuelidle = fuelusage[0]

self.fuelACV = fuelusage[1]

def run(self, j, ISBstores, SBstores, beachemptystores):

"""The PEM for AirCushionVehicles. This method overrides the Connector run method.

The ACVs start at the ISB with no cargo and are transported to the Sea

Base

aboard another vessel, such as the MLP. If the Sea Base is positioned within the range of the ACV then the ACV serves as a basic connector, following the

general procedure for a connector.

If the Sea Base is positioned beyond the range of the ACV, the MLP must be

used to ferry the ACVs to some standoff distance. Instead of calculating the

distance to travel and waiting that amount of time, the ACVs must wait for an

MLP to carry them, forming a queue. Once the MLP reaches the standoff distance,

it offloads and the ACVs travel the remaining distance to shore and find a landing zone on the beach. Once unloaded, the ACVs repeat the same process with

the ACVs queuing to be ferried by the MLP and unloaded upon arrival at the Sea Base,

joining the loading queue.

"""

namematrix(self)

yield hold, self, .25

##Put self into queue to be carried from ISB

yield put, self, ISBstores, [self]

yield waitevent, self, self.SBSignal

```
self.location = "SBstart"
hasgoal = choosegoal(self)

while not hasgoal:
    yield waitevent,self,checkvessel
    hasgoal = choosegoal(self)

while True:
    time1 = now()
    hascargoandinterface = False
    while not hascargoandinterface:
        if self.purpose == "pull":
            [schedule, priority] = idschedule(self, self.nextpush)
            self.cargowant, loadoutarea, loadoutwt, relativepriority = choose-
cargo(self, schedule, priority, self.goal)
        cargoatlocation = calcloc(self, self.goal)
        spotswithcargo = checkspotcargo(self, cargoatlocation)
        hascargoandinterface = num.amax(spotswithcargo)

        if not hascargoandinterface:
            yield waitevent,self,checkvessel

    row, col, wait, incomp = findspot(self, spotswithcargo, j)
    if self.purpose == "pull":
        self.loadareas.append(loadoutarea)
        self.loadwts.append(loadoutwt)
        updateschedule(self)
```

```
timequeued = now() - time1
self.repaircount += timequeued
self.fuel += timequeued * self.fuelidle
if self.goal.startswith("SB"):
    self.loadqueue += timequeued
    if self.repaircount >= self.MTBF:
        yield hold, self, self.MTTR
        self.repaircount = 0
else:
    self.unloadqueue += timequeued

yield hold, self, wait
self.load += wait

if self.goal.strip("0123456789") in self.pushlocs or self.goal in self.pushlocs:
    self.numunloaded += 1

Spots[row, col] = 1
Spots[row] = Spots[row] + incomp

hasgoal = self.changemission(row, j)

while not hasgoal:
    yield waitevent,self,checkvessel
    hasgoal = choosegoal(self)
```

```
                checkvessel.signal()
                yield hold, self, 0

                dist = Distances[self.location][self.goal]
                strippedname = self.name.strip("0123456789")
                if self.vehiclerange < dist: #if ACV can not travel to shore on its own,
it must wait and be ferried closer to shore
                    if self.location.startswith("SB"): #going to beach

                        time2 = now()
                        yield put, self, globals()[self.goal+'SB'+strippedname], [self]
                        yield waitevent, self, self.arrivedSignal
                        self.MLPqueueSB += now() - time2
                        self.fuel += (now() - time2) * self.fuelidle
                        yield hold, self, self.vehiclerange / self.speed * 60
                        self.travel += self.vehiclerange / self.speed * 60
                        self.repaircount += self.vehiclerange / self.speed * 60
                        self.fuel += self.vehiclerange / self.speed * 60 * self.fuelACV
                    else:
                        yield hold, self, self.vehiclerange / self.speed * 60
                        self.travel += self.vehiclerange / self.speed * 60
                        self.repaircount += self.vehiclerange / self.speed * 60
                        self.fuel += self.vehiclerange / self.speed * 60 * self.fuelACV
                        time2 = now()
                        yield put, self, globals()[self.location+'empty'+strippedname], [self]

                        yield waitevent, self, self.unloadedSignal
```

```
                self.MLPqueuebeach += now() - time2
                self.fuel += (now() - time2) * self.fuelidle
            else: #if within its range, ACV will travel on its own
                dist = Distances[self.location][self.goal]
                yield hold, self, (dist / self.speed * 60)
                self.travel += dist / self.speed * 60
                self.repaircount += dist / self.speed * 60
                self.fuel += dist / self.speed * 60 * self.fuelACV


    class LCAC(AirCushionVehicle):
        """Subclass of AirCushionVehicle for the LCAC."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, vehiclerange,
fuelusage, hopping):
            AirCushionVehicle.init(self, name, spots, wait, maxlift, lifteff, maxarea,
areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, vehiclerange,
fuelusage,[], hopping)
            LCAC.roster.append(self)


    class LCACR(AirCushionVehicle):
        """Subclass of AirCushionVehicle for the LCAC-R."""
        roster = []
        def init(self, name, spots, wait, maxlift, lifteff, maxarea, areaeff,
                 MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, vehiclerange,
```

```
fuelusage, hopping):
        AirCushionVehicle.init(self, name, spots, wait, maxlift, lifteff, maxarea,
areaeff,
                MTBF, MTTR, compatibility, speed, pushlocs, pulllocs, vehiclerange,
fuelusage,[], hopping)
        LCACR.roster.append(self)

    class SBCargoShip(Process):
        """General Process for Sea Base cargo ships.

        Cargo ships act as cargo objects that serve to interface with connectors and
hold cargo.
        They must start at some initial starting location and travel to the Sea Base or
other
        cargo spot, such as a nearby port. While a cargo object is stationary, no pro-
cess
        definition is needed beyond the definition of the initial properties.

        Implemented by AmphibiousShip.
        """
        def init(self, name, spots, cargo, speed):
            Process.init(self)
            self.name = name
            self.spots = spots
            self.cargohave = cargo
            self.speed = speed
            self.cargowant = num.zeros(numcargo)
```

```
            self.goal = "SB"
            self.purpose = ""
            self.type = "in transport"
            self.wait = num.zeros(numinterfaces)

        def deploy(self, vehiclelist):
            """Initial assignment of vessels to start locations.  Overrides Connector
deploy method."""
            vehicleid = len(vehiclelist) - 1
            if vehicleid%numISB == 0:
                locationid = 0
            else:
                vehicle = vehiclelist[-2]
                previousid = vehicle.location[-2:]
                locationid = int(previousid) + 1
            self.location = "ISB%02d" %locationid


    class AmphibiousShip(SBCargoShip):
        """Subclass of SBCargoShip for amphibious warfare ships.

        Distinguished by its ability to carry helicopters and smaller vessels in its well
deck.
        This ship ferries these objects to the SB, where they become individual con-
nectors and
        then becomes a resupply point.

        Implemented by LMSR, LHD, LPD, LSD
```

```
    """
    def init(self, name, spots, cargo, speed, helospots, LCspots, INLSspots):
        SBCargoShip.init(self, name, spots, cargo, speed)
        self.Heloid = []
        self.LCid = []
        self.INLSid = []
        self.LCspots = LCspots
        self.helospots = helospots
        self.INLSspots = INLSspots

    def run(self, j):
        """The PEM for AmphibiousShip. This method overrides the Connector
run method.

        This ship ferries these objects to the SB, where they become individual
connectors and
        then becomes a resupply point. The number of each of helicopter and
smaller ship that
        can be carried are set as properties, but only a single type is selected.
        """
        namematrix(self)
        #Travel to ISB
        yield hold, self, 5
        #Load helos
        self.NumcarriedHelo = min(ISBCH46.nrBuffered, self.helospots[0])
        self.NumcarriedHelo2 = min(ISBCH53.nrBuffered, self.helospots[1])
        if self.NumcarriedHelo > self.NumcarriedHelo2: #Select helos to carry by
```

```
selecting max number
                Helochoice = 1
            else:
                Helochoice = 2
            if Helochoice == 1: #Load selected helos
                for r in range(self.NumcarriedHelo):
                    yield get, self, ISBCH46, 1
                    whichHelo = self.got[0]
                    self.Heloid.append(whichHelo)
            if Helochoice == 2:
                for r in range(self.NumcarriedHelo2):
                    yield get, self, ISBCH53, 1
                    whichHelo = self.got[0]
                    self.Heloid.append(whichHelo)
            #Load landing craft
            self.NumcarriedLCAC = min(ISBLCAC.nrBuffered, self.LCspots[0])
            self.NumcarriedLCACR = min(ISBLCACR.nrBuffered, self.LCspots[0])
            self.NumcarriedLCU1600 = min(ISBLCU1600.nrBuffered, self.LCspots[1])
            templist = [self.NumcarriedLCAC,self.NumcarriedLCACR,self.NumcarriedLCU1600]
            LCchoice = templist.index(max(templist))
            if LCchoice == 0:
                for r in range(self.NumcarriedLCAC): #reserve loaded LCACs till have
max can carry
                    yield get, self, ISBLCAC, 1
                    whichLCAC = self.got[0]
                    self.LCid.append(whichLCAC)
            elif LCchoice == 1:
```

```
            for r in range(self.NumcarriedLCACR): #reserve loaded LCACRs till
have max can carry
                yield get, self, ISBLCACR, 1
                whichLCACR = self.got[0]
                self.LCid.append(whichLCACR)
        elif LCchoice == 2:
            for r in range(self.NumcarriedLCU1600): #reserve loaded LCU1600s
till have max can carry
                yield get, self, ISBLCU1600, 1
                whichLCU1600 = self.got[0]
                self.LCid.append(whichLCU1600)
        #Load INLS
        self.NumcarriedINLS = min(ISBINLS.nrBuffered, INLSspots)
        for r in range(self.NumcarriedINLS): #load INLS
            yield get, self, ISBINLS, 1
            whichINLS = self.got[0]
            self.INLSid.append(whichINLS)
        #Travel to Sea Base
        dist = Distances[self.location][self.goal]
        traveltime = dist/self.speed*60
        yield hold, self, traveltime
        self.location = self.goal

        #Release helos, landing craft, INLS
        if Helochoice == 1: #Release helos
            for s in range(self.NumcarriedHelo):
                whichHelo = self.Heloid.pop()
```

```
            whichHelo.SBSignal.signal()
            yield hold, self, 0
    elif Helochoice == 2:
        for s in range(self.NumcarriedHelo2):
            whichHelo = self.Heloid.pop()
            whichHelo.SBSignal.signal()
            yield hold, self, 0
    if LCchoice == 0: #Release landing craft
        for s in range(self.NumcarriedLCAC):
            whichLCAC = self.LCid.pop()
            whichLCAC.SBSignal.signal()
            yield hold, self, 0
    elif LCchoice == 1:
        for s in range(self.NumcarriedLCACR):
            whichLCACR = self.LCid.pop()
            whichLCACR.SBSignal.signal()
            yield hold, self, 0
    elif LCchoice == 2:
        for s in range(self.NumcarriedLCU1600):
            whichLCU1600 = self.LCid.pop()
            whichLCU1600.SBSignal.signal()
            yield hold, self, 0
    for s in range(self.NumcarriedINLS): #release INLS
        whichINLS = self.INLSid.pop()
        whichINLS.SBSignal.signal()
        yield hold, self, INLSTimeToOffload
    self.goal = ""
```

```
            self.purpose = ""
            self.type = "cargo"
            replacematrixcargoship(self, j) #Becomes a cargo ship at Sea Base
            checkvessel.signal()
            yield hold, self, 0


    class LMSR(AmphibiousShip):
        """Subclass of AmphibiousShip for the LMSR."""
        roster = []
        def init(self, name, spots, cargo, speed, helospots, LCspots):
            AmphibiousShip.init(self, name, spots, cargo, speed, helospots, LCspots,
INLSspots = 0)
            LMSR.roster.append(self)
            self.deploy(LMSR.roster)


    class LHD(AmphibiousShip):
        """Subclass of AmphibiousShip for the LHD."""
        roster = []
        def init(self, name, spots, cargo, speed, helospots, LCspots):
            AmphibiousShip.init(self, name, spots, cargo, speed, helospots, LCspots,
INLSspots = 1)
            LHD.roster.append(self)
            self.deploy(LHD.roster)
            self.goal = "SB"#"SBclose"


    class LPD(AmphibiousShip):
        """Subclass of AmphibiousShip for the LPD."""
```

```
        roster = []
        def init(self, name, spots, cargo, speed, helospots, LCspots):
            AmphibiousShip.init(self, name, spots, cargo, speed, helospots, LCspots,
INLSspots = 1)
            LPD.roster.append(self)
            self.deploy(LPD.roster)

    class LSD(AmphibiousShip):
        """Subclass of AmphibiousShip for the LSD."""
        roster = []
        def init(self, name, spots, cargo, speed, helospots, LCspots):
            AmphibiousShip.init(self, name, spots, cargo, speed, helospots, LCspots,
INLSspots = 1)
            LSD.roster.append(self)
            self.deploy(LSD.roster)

    class MLP(Process):
        """Generic process for MLP. Completely separate from the Connector process.

        The MLP is a special case as it serves as a cargo transfer enabler as well
as
        having the potential to disconnect from the LMSR and travel as an indepen-
dent
        object. In the case where the MLP stays attached to the LMSR and does not
ferry
        ACVs, the MLP simply serves to enable a loading connection on the LMSR.
        """
```

```
roster = []
def init(self, name):
    Process.init(self)
    self.name = name
    self.cargohave = num.zeros(numcargo)
    self.cargowant = num.zeros(numcargo)
    self.goal = "SB"
    self.purpose = "pull"
    self.type = "MLP"
    self.shipid = []
    self.carry = ""
    MLP.roster.append(self)
    self.deploy(MLP.roster)

def MLP(self, j, MLPloading):
    """The PEM for MLP.

    The MLP starts at the ISB and travels to the Sea Base.

    If there are no ACVs, the MLP identifies an available LMSR and the makes
a connection.
    In this way, the MLP becomes a part of the LMSR and is not considered
a separate object.

    On the other hand, when the MLP is used to ferry ACVs, it starts unattached
and
    immediately looks for ACVs to load from the queue of loaded objects at
```

the Sea Base.

Once the connectors are on board, the MLP travels to the standoff distance from

the shore. On arrival, it triggers the reactivation of the ACVs so they can travel to shore and unload. The MLP waits at the standoff distance for the return of the ACVs, and once they are on board, returns to the Sea Base.

"""

```
def checkready():
    shipwaiting = False
    for beach in beaches:
        if globals()[beach+'SBLCAC'].nrBuffered >= LCACperMLP or globals()[beach+'SBLCACR'].nrBuffered >= LCACRperMLP:
            shipwaiting = True
    return shipwaiting


self.spots = MLPspots
self.wait = MLPconn
namematrix(self)
yield hold, self, 2
##ISB to SB, load ACVs
self.Numcarried = min(ISBLCAC.nrBuffered, LCACperMLP)
self.NumcarriedR = min(ISBLCACR.nrBuffered, LCACRperMLP)


if ISBLCAC.nrBuffered >= 1:
    for r in range(self.Numcarried): #reserve loaded LCACs till have max can carry
        yield get, self, ISBLCAC, 1
```

```
                whichLCAC = self.got[0]
                self.shipid.append(whichLCAC)
            self.carry = "LCAC"
            MLPonload = MLPonloadLCAC
        elif ISBLCACR.nrBuffered >= 1:
            for r in range(self.NumcarriedR): #reserve loaded LCACRs till have
max can carry
                yield get, self, ISBLCACR, 1
                whichLCACR = self.got[0]
                self.shipid.append(whichLCACR)
            self.carry = "LCACR"
            MLPonload = MLPonloadLCACR
        else:
            MLPonload = 0
            self.carry = ""

        # if either of nrBuffered is non-zero, hold the process.
        if ISBLCAC.nrBuffered + ISBLCACR.nrBuffered:
            yield hold, self, MLPonload

        MLP.fuel += MLPonload*MLPfuelidle

        dist = Distances[self.location][self.goal] #Travel to Sea Base
        traveltime = dist/MLPspeed*60
        yield hold, self, traveltime

        MLP.travel += traveltime
```

```
MLP.fuel += traveltime*MLPfuelusage
self.location = "SB"

if self.carry == "LCAC": #Release ACVs
    MLPoffload = MLPoffloadLCAC
    numcarried = self.Numcarried
    vehiclelist = self.shipid
elif self.carry == "LCACR":
    MLPoffload = MLPoffloadLCACR
    numcarried = self.NumcarriedR
    vehiclelist = self.shipid
elif self.carry == "":
    MLPoffload = 0
    numcarried = 0

if self.carry != "":
    yield hold, self, MLPoffload

MLP.fuel += MLPoffload*MLPfuelidle
for s in range(numcarried):
    vehicle = self.getvehicle(vehiclelist)
    vehicle.SBSignal.signal()
    yield hold, self, 1

self.carry = ""
replacematrixcargoship(self, j)
```

```
while True:
    yield request, self, MLPloading
    conntoload = checkready()
    if conntoload ==True:
        beachtoload, shiptype = self.choosebeach()
        for r in range(eval(shiptype+'perMLP')):
            yield get, self, globals()[beachtoload+'SB'+ shiptype], 1
            whichship = self.got[0]
            self.shipid.append(whichship)
            self.carry = shiptype
            self.goal = beachtoload


    else:
        yield release, self, MLPloading
        self.goal = "SB"
        time3 = now()
        hascargoandinterface = False
        while not hascargoandinterface:
            cargoatlocation = calcloc(self, self.goal)
            spotswithcargo = checkspotcargo(self, cargoatlocation)
            hascargoandinterface = num.amax(spotswithcargo)
            if not hascargoandinterface:
                yield waitevent, self, checkvessel
        row, col, wait, incomp = findspot(self, spotswithcargo, j)
        time4 = now() - time3
        MLP.SBqueue += time4
        MLP.fuel += time4*MLPfuelidle
```

```
                yield hold, self, wait
                Spots[row, mlpspot] = 1
                allSpots[row, mlpspot] = 1
                time5 = now()

                checkvessel.signal()
                yield hold, self, 0

                ##take lcac or lcacR?
                yield waitevent, self, checkvessel
                yield request, self, MLPloading #want to load LCACs onto 1 MLP
to it can leave
                conntoload = checkready()
                while conntoload == False:
                    yield release, self, MLPloading
                    yield waitevent, self, checkvessel
                    yield request, self, MLPloading #want to load LCACs onto 1
MLP to it can leave
                    conntoload = checkready()

                beachtoload, shiptype = self.choosebeach()
                for r in range(eval(shiptype+'perMLP')):
                    yield get, self, globals()[beachtoload+'SB'+ shiptype], 1
                    whichship = self.got[0]
                    self.shipid.append(whichship)
                    self.carry = shiptype
                    self.goal = beachtoload
```

```
        yield release, self, MLPloading
        time6 = now() - time5
        #MLP.SBqueue += time6
        Spots[row, mlpspot] = 0

        yield hold, self, MLPdisconn
        Spots[row, col] = 1
        Spots[row] = Spots[row] + incomp

    #create for LCAC and LCACR
    if self.carry == "LCAC":
        MLPonload = MLPonloadLCAC
        vehiclerange = LCACrange
        MLPoffload = MLPoffloadLCAC
        vehicleperMLP = LCACperMLP
        vehiclelist = self.shipid
    elif self.carry == "LCACR":
        MLPonload = MLPonloadLCACR
        vehiclerange = LCACRrange
        MLPoffload = MLPoffloadLCACR
        vehicleperMLP = LCACRperMLP
        vehiclelist = self.shipid
    elif self.carry == "":
        MLPonload = 0

    if self.carry != "":
```

```
    yield hold, self, MLPonload
MLP.fuel += MLPonload*MLPfuelidle
self.location = "SB"
time1 = now()

yield hold, self, (Distances[self.location][self.goal] - vehiclerange)/MLPspeed*60
MLP.fuel += (Distances[self.location][self.goal] - vehiclerange)/MLPspeed*60*MLPfu
yield hold, self, MLPoffload
MLP.fuel += MLPoffload*MLPfuelidle
##process to transport LCACs
time2 = now() - time1

if self.carry == "LCAC":
    LCAC.MLPqueueSB = sum([lcac.MLPqueueSB for lcac in LCAC.roster])
    LCAC.MLPqueueSB -= time2*LCACperMLP
else:
    LCACR.MLPqueueSB = sum([lcacr.MLPqueueSB for lcacr in LCACR.roster])
    LCACR.MLPqueueSB -= time2*LCACRperMLP
self.location = self.goal
self.goal = "SB"
for s in range(vehicleperMLP):
    vehicle = self.getvehicle(vehiclelist)
    vehicle.arrivedSignal.signal()
    yield hold, self, 0
time5 = now()
for r in range(vehicleperMLP):
    yield get, self, globals()[self.location+'empty'+self.carry], 1
```

```
            vehicle = self.got[0]
            self.putvehicle(vehiclelist, vehicle)
        time6 = now() - time5
        MLP.beachqueue += time6
        MLP.fuel += time6*MLPfuelidle
        time1 = now()
        yield hold, self, MLPonloadLCAC #time to onload LCAC
        MLP.fuel += MLPonloadLCAC*MLPfuelidle
        dist = Distances[self.location][self.goal]
        yield hold, self, (dist - vehiclerange)/MLPspeed*60
        MLP.fuel += (dist - vehiclerange)/MLPspeed*60*MLPfuelusage
        yield hold, self, MLPoffload
        MLP.fuel += MLPoffload*MLPfuelidle
        time2 = now() - time1

        if self.carry == "LCAC":
            LCAC.MLPqueuebeach = sum([lcac.MLPqueuebeach for lcac in LCAC.roster])
            LCAC.MLPqueuebeach -= time2*LCACperMLP
        else:
            LCACR.MLPqueuebeach = sum([lcacr.MLPqueuebeach for lcacr in
LCACR.roster])
            LCACR.MLPqueuebeach -= time2*LCACRperMLP

        for s in range(vehicleperMLP):
            vehicle = self.getvehicle(vehiclelist)
            vehicle.unloadedSignal.signal()
            yield hold, self, 0
```

```
def getvehicle(self, array):
    """Remove and return the last item of the given array."""
    return array.pop()

def putvehicle(self, array, vehicle):
    """Append the given item to the end of the given array."""
    array.append(vehicle)

def deploy(self, vehiclelist):
    """Initial assignment of vessels to start locations."""
    vehicleid = len(vehiclelist) - 1
    if vehicleid%numISB == 0:
        locationid = 0
    else:
        vehicle = vehiclelist[-2]
        previousid = vehicle.location[-2:]
        locationid = int(previousid) + 1
    self.location = "ISB%02d" %locationid

def choosebeach(self):
    beachtoload = []
    for beach in beaches:
        if globals()[beach+'SBLCAC'].nrBuffered >= LCACperMLP:
            beachtoload = beach
            shiptype = "LCAC"
        elif globals()[beach+'SBLCACR'].nrBuffered >= LCACRperMLP:
```

```
            beachtoload = beach
            shiptype = "LCACR"

        return beachtoload, shiptype

    class ISB(Process):
        """Generic process for the intermediary staging base.

        The intermediary staging base (ISB) is a base where the assets are pre-positioned
and a
        point of resupply. These bases are at a set distance from the theater with an
inputted
        amount of cargo available. These bases can have up to three piers for resupply.
        """
        def init(self, name):
            Process.init(self)
            self.name = name

            ISBcargohave = eval(self.name + 'cargohave')
            ISBspots= eval(self.name + 'spots')
            self.spots = ISBspots
            self.cargohave = ISBcargohave
            self.cargowant = num.zeros(numcargo)
            self.location = self.name
            self.goal =""
            self.purpose=""
            self.type = "cargo"
```

```
        self.wait = num.zeros(numinterfaces)

    def run(self):
        namematrix(self)
        yield hold, self, 0

class Port(Process):
    """Generic process for the Port.

    This represents a port near the theater of interest that can be used as a throughput point.
    This port does not have any cargo at the start of the simulation but serves as a pier where
    cargo can be offloaded from larger ships that are either faster to offload at a pier or can
    not transfer cargo at sea. This cargo is then loaded onto smaller connectors that transfer
    it directly to the beach.
    """
    def init(self, name):
        Process.init(self)
        self.name = name
        global Portcargowant
        self.spots = Portspots
        self.cargohave = num.zeros(numcargo)
        self.cargowant = num.zeros(numcargo)
        self.location = self.name
```

```
        self.goal =""
        self.purpose=""
        self.type = "cargo"
        self.wait = num.zeros(numinterfaces)

    def run(self, j):
        namematrix(self)
        replacematrix(self, j)
        yield hold, self, 0

class Shore(Process):
    """Generic process for the Shore.

    Beaches are established as a stationary object by defining the initial properties and
    no additional process definition is needed. The different types of landing zones are
    identified by the interfaces available and those must match the designated connector type.
    """
    def init(self, name, spots, cargowant, location):
        Process.init(self)
        self.name = name
        self.wait = num.zeros(numinterfaces)
        self.cargohave = num.zeros(numcargo)
        self.spots = spots
        self.cargowant = cargowant
```

```
        self.goal = ""
        self.purpose = ""
        self.type = "cargo"
        self.location = location

    def run(self):
        namematrix(self)
        yield hold, self, 0

class ShoreBeach(Shore):
    """Subclass of Shore for beaches."""
    def run(self):
        for beach in BeachTypes:
            for beachtype in beaches:
                for n in range(BeachTypes[beach][beachtype]):
                    name = beach +'%02d'%n
                    location = beachtype
                    spots = eval(beach + 'spots')
                    cargowant = eval(beach + 'cargowant')
                    c = Shore(name, spots, cargowant, location)
                    activate(c, c.run())
                    yield hold, self, 0

## General scripts ----------

def namematrix(self):
    """Add properties of an object to the overall matrices."""
```

```
        Names.append(self.name)
        Location.append(self.location)
        Goal.append(self.goal)
        Spots2.append(self.spots)
        global Spots
        Spots = num.array(Spots2)
        global allSpots
        allSpots = num.array(Spots2)
        Predictedwaits2.append(num.zeros(numinterfaces))
        global Predictedwaits
        Predictedwaits = num.array(Predictedwaits2)
        Predictedcargo2.append(num.zeros(numcargo))
        global Predictedcargo
        Predictedcargo = num.array(Predictedcargo2)
        Cargohave2.append(self.cargohave)
        global Cargohave
        Cargohave = num.array(Cargohave2)
        Cargowant2.append(self.cargowant)
        global Cargowant
        Cargowant = num.array(Cargowant2)
        Type2.append(self.type)
        global Type
        Type = num.array(Type2)

    def calcloc(self, location):
        """Identify which cargo objects are at the given location.
```

```
        Returns an array with entries of one or zero for each object:
        -one signifies a cargo object that is at the given location
        -zero signifies not a cargo object or not at the given location.
        """
        locationvector = num.array([int(L == location) for L in Location])
        typevector = num.array([int(T == "cargo") for T in Type])
        cargoatlocation = locationvector*typevector
        return cargoatlocation

    def calcgoal(self, goal):
        """Identify connector objects with a given goal, i.e. traveling to a given location.

        Returns an array with entries of one or zero for each object:
        -one signifies a connector object with the given goal
        -zero signifies not a connector object or doesn't have the given goal.
        """
        goalvector = num.array([int(G == goal) for G in Goal])
        typevector = num.array([int(T == "connector") for T in Type])
        connectorwithgoal = goalvector*typevector
        return connectorwithgoal

    def Portcargoneeds(self):
        """Returns the port cargo schedule with non-negative values."""
        return [max(ps, 0) for ps in Portcargowant]

    def choosecargo(self, Schedule, Priority, pulllocation):
```

"""Identify cargo to load using given schedule and priority arrays.

Creates and solves the linear programming problem:

-maximize the value of cargo carried, value = (normalizedweight + normalizedarea+1) / priority

-linear constraints are the lift and weight of each cargo item

-side contraints are lift and weight capacity of the connector object

-Additional side constraints for limiting cargo selection to one cargo supply point/ vessel

-lower bound is an array of no cargo

-upper bound is an array of the desired cargo

Returns:

-cargowant: an array with entries for each cargo item designating amount to carry

-loadoutarea: a float for the total area of the cargo

-loadoutwt: a float for the total weight of the cargo

-relativepriority: a float for the total value of prioritized cargo

"""

```
global Cargohave
global numships
Cargo = [i+1 for i in range(numcargo)]
cargovalue = num.zeros(numcargo)
avgwt = num.average(cargowts)
avgarea = num.average(cargoarea)
normalizedwtarea = cargowts/avgwt + cargoarea/avgarea
#Prioritization of cargo... if smaller things are as important, it
```

```
        #will be sent first, so weighted by area and weight of cargo item

        cargovalue = [(nwa + 1)/p for nwa, p in zip(normalizedwtarea, Priority)]
        f = num.array(num.concatenate((cargovalue, num.zeros(numships)), axis=1))
        cargodesired = Schedule * self.compatibility

        cargowts2 = num.concatenate((cargowts, num.zeros(numships)), axis=1)
        cargoarea2 = num.concatenate((cargoarea, num.zeros(numships)), axis=1)
        cargolocation = calcloc(self, pulllocation)
        cargoloc2 = []
        for m in range(numcargo):
            cargoloc2.append(cargolocation)
        cargoavail = Cargohave * num.transpose(cargoloc2)
        cargoinput = num.concatenate((num.identity(numcargo), - num.transpose(cargoavail)),
axis=1)
        selectvessel = num.concatenate((num.zeros(numcargo), num.ones(numships)))

        a = num.concatenate(([cargowts2, cargoarea2, selectvessel], cargoinput))
        b = num.concatenate(([self.lift, self.area,1], num.zeros(numcargo)), axis=0)
        vlb = num.concatenate((num.zeros(numcargo), num.zeros(numships)), axis=0)
        vub = num.concatenate((cargodesired, num.ones(numships)), axis=0)

        totalnumvar = numcargo + numships
        allvariables = num.zeros(totalnumvar)
        for i in range(totalnumvar):
            allvariables[i] = i + 1
        xint = allvariables
```

```
        e = num.concatenate(([-1,-1,0], -1*num.ones(numcargo)), axis=0)
        [obj, x, duals] = lpsolve(f, a, b, e, vlb, vub, xint)

        cargowant1=num.array(x)
        cargowant=cargowant1[0:numcargo]
        loadoutwt = num.sum(cargowts * cargowant)
        loadoutarea = num.sum(cargoarea * cargowant)
        if num.max(cargowant) == 0: #if no cargo is selected, set to infinate so the
connector does not move forward
            cargowant = num.inf * num.ones(numcargo)
        relativepriority = num.sum(cargovalue * cargowant)

        return cargowant, loadoutarea, loadoutwt, relativepriority

    def choosehops(self, pulllocation, Predictive):
        """Identify cargo to load and beaches to visit if multiple unloads (hopping) is
an option.

        Creates and solves the linear programming problem:
        -maximize the value of cargo carried, value = (normalizedweight + normal-
izedarea+1) / priority
        While minimizing total trip time, normalized by shortest possible trip
        -linear constraints are the lift and weight of each cargo item
        -side contraints are lift and weight capacity of the connector object
        -Assignment side constraints for limiting cargo selection to one cargo supply
point/ vessel
        -Traveling salesman side constraints to determine beaches to visit
```

```
        -Must start and stop at beach 0, a dummy placeholder
        -No subtours
    -max cargo based on demand at beaches selected

    Returns:
    -cargowant: an array with entries for each cargo item designating amount to
carry
    -loadoutarea: a float for the total area of the cargo
    -loadoutwt: a float for the total weight of the cargo
    -relativepriority: a float for the total value of prioritized cargo
    -beachorder: order of beaches to visit (numerical)
    -beachnames: order of beaches to visit (names)
    -totaltriptime: a float for the predicted total trip time
        -travel, wait, load, travel, unload, travel if more than 1 beach...
    """
    global Cargohave
    global numships

    #Calculate Demand matrix
    cargolocation = calcloc(self, pulllocation)
    cargodesired = num.zeros((numcargo))
    numbeaches = 1 + len(beaches) #beach 0 is placeholder
    traveltimecalc = num.zeros(numbeaches)
    beachcount = 1
    for beach in beaches:
        if Predictive == True:
            tempschedule, Priority = idpredictedschedule(self,beach)
```

```
                self.cargowant = num.zeros(numcargo)
                findtraveltime = Distances[pulllocation][beach]/ self.speed * 60
                traveltimecalc[beachcount] = findtraveltime
                beachcount +=1
            else:
                tempschedule, Priority = idschedule(self,beach)

            cargodesired = num.vstack((cargodesired,tempschedule))

        cargodesired = cargodesired * self.compatibility

        #Identify push spot queue times
        beachcostwait = num.zeros( (len(beaches)+1, len(beaches)+1))
        beachcostunload = num.zeros( (len(beaches)+1, len(beaches)+1))
        beachcosttravel = num.zeros( (len(beaches)+1, len(beaches)+1))
        beachnum = 0
        cargoatlocation = calcloc(self, pulllocation)
        for beach in beaches:
            beachnum += 1
            cargoatlocation = calcloc(self, beach)
            pushcompatiblespots = checkallspots(self, cargoatlocation, "push")
            loadingtimes = pushcompatiblespots * self.wait
            queueingtimes = pushcompatiblespots * Predictedwaits
            waits = loadingtimes + queueingtimes
            waits[waits <= 0] = num.inf # zeros are replaced with infinity before at-
tempting to find the minimum waiting time
            shortestrow, shortestcol = num.unravelindex(waits.argmin(), waits.shape)
```

```
            pushloadingtime = waits[shortestrow, shortestcol]
            pushwaittime = queueingtimes[shortestrow, shortestcol]

            #store push data
            globals()['shortestcol'+beach] = shortestcol
            globals()['shortestrow'+beach] = shortestrow
            globals()['loadingtimes'+beach] = loadingtimes
            #apply compatibility
            Taken = num.zeros(numinterfaces)
            Taken[shortestcol] = 1
            Incspots = Incompatible * Taken
            globals()['incrow'+beach] = Incspots.sum(1)
            beachnum1 = 0

            for beach1 in beaches:
                beachnum1 += 1
                beachcostunload[:,beachnum] = pushloadingtime
                beachcostwait[beachnum1,beachnum] = pushwaittime
                beachcosttravel[beachnum1,beachnum] = Distances[beach][beach1]/ self.speed
* 60
                beachcosttravel[0, beachnum1] = traveltimecalc[beachnum1]

        beachcostcalc = num.maximum(beachcosttravel, beachcostwait) + beachcos-
tunload

        Cargo = [i+1 for i in range(numcargo)]
        cargovalue = num.zeros(numcargo)
```

```
        avgwt = num.average(cargowts)
        avgarea = num.average(cargoarea)
        normalizedwtarea = cargowts/avgwt + cargoarea/avgarea
        #Prioritization of cargo... if smaller things are as important, it
        #will be sent first, so weighted by area and weight of cargo item

        cargovalue = [1*(nwa + 1)/p for nwa, p in zip(normalizedwtarea, Priority)]

        beachcosts2 = num.array([item for innerlist in beachcostcalc for item in in-
nerlist ])
        beachcosts2[beachcosts2 <= 0] = num.amax(beachcosts2) # zeros are replaced
with infinity before attempting to find the minimum waiting time
        mincost = num.average(beachcosts2)
        beachcosts = [-1*item/mincost for innerlist in beachcostcalc for item in in-
nerlist ]

        f = num.array(num.concatenate((cargovalue, num.zeros(numships), num.zeros(numbeaches),
beachcosts), axis=1))

        cargowts2 = num.concatenate((cargowts, num.zeros(numships + (numbeaches*(numbeaches-
axis=1)
        cargoarea2 = num.concatenate((cargoarea, num.zeros(numships+ (numbeaches*(numbeaches
axis=1)

        cargoloc2 = []
        for m in range(numcargo):
            cargoloc2.append(cargolocation)
```

```
        cargoavail = Cargohave * num.transpose(cargoloc2)
        cargoinput2 = num.concatenate((num.identity(numcargo), - num.transpose(cargoavail)),
axis=1)
        cargoinput = num.concatenate((cargoinput2, num.zeros([numcargo, numbeaches*(numbeach
axis=1)
        selectvessel2 = num.concatenate((num.zeros(numcargo), num.ones(numships)))
        selectvessel = num.concatenate((selectvessel2,num.zeros(numbeaches + (num-
beaches*numbeaches))))

        #set demand
        demandinput4 = num.concatenate((num.identity(numcargo), num.zeros([numcargo,numship
axis=1)
        demandinput3 = num.concatenate((-num.transpose(cargodesired), num.zeros([numcargo,nu
axis=1)
        demandinput = num.concatenate((demandinput4, demandinput3),axis=1)
        #set route selection
        beachidentity = num.identity(numbeaches)
        beachmatrix=beachidentity
        for n in range(numbeaches-1):
            beachmatrix = num.concatenate((beachmatrix,beachidentity),axis=1)
        beachmatrix = num.concatenate((num.zeros([numbeaches, numcargo + numships+
numbeaches]), beachmatrix), axis=1)

        #must visit each beach at most once
        beachmatrix1=num.zeros([numbeaches,numbeaches*numbeaches])
        for m in range(numbeaches):
            for p in range(numbeaches*numbeaches):
```

```
                if (p-(m+1)*numbeaches)<=-1 and (p-(m)*numbeaches)>=0:
                    beachmatrix1[m,p] = 1
        beachmatrix2 = num.concatenate((num.zeros([numbeaches, numcargo + numships+
numbeaches]), beachmatrix1), axis=1)

        #Diagonal of nm,p must be 0
        beachmatrix3 = num.zeros([numbeaches,numbeaches*numbeaches])
        for m in range(numbeaches):
            for p in range(numbeaches*numbeaches):
                if (p-m*numbeaches)==m:
                    beachmatrix3[m,p] = 1
        beachmatrix3 = num.concatenate((num.zeros([numbeaches, numcargo + numships+
numbeaches]), beachmatrix3), axis=1)

        subtour1 = num.tri(numbeaches, numbeaches, k=0)
        subtour4 = num.zeros((numbeaches, numbeaches))
        for r in range(numbeaches):
            for m in range(numbeaches):
                    if r != 0 and m == 0:
                        subtour4[r,m] = 1
        subtour5 = subtour1 - subtour4 - num.identity(numbeaches)
        subtour = num.array([item for innerlist in subtour5 for item in innerlist ])

        beachmatrix4 = num.zeros([((numbeaches - 2)*(numbeaches - 1))/2, num-
beaches*numbeaches])
        for m in range((((numbeaches - 2)*(numbeaches - 1))/2)):
            matrixform = num.zeros((numbeaches,numbeaches))
```

```
        check = subtour
        vector = num.zeros(num.shape(subtour))
        for p in range(numbeaches*numbeaches):
            if check[p] ==1:
                vector[p] = 1
                matrix1 = num.reshape(vector, (numbeaches,-1))
                for r in range(numbeaches):
                    for q in range(numbeaches):
                        if matrix1[r,q] == 1:
                            matrixform[q,r]=1
                matrixform = matrixform + matrix1
                subtour[p] = 0
                check = num.zeros(num.shape(subtour))
    beachmatrix4[m:] = num.array([item for innerlist in matrixform for item
in innerlist ])
    beachmatrix4 = num.concatenate((num.zeros([((numbeaches - 2)*(numbeaches
- 1))/2, numcargo + numships+ numbeaches]), beachmatrix4), axis=1)

    #set start and end at dummy beach (beach 0)
    startmatrix = num.vstack((beachmatrix[0,],beachmatrix2[0,]))

    #start and end must be from active beaches
    beachmatrix5 = num.zeros((numbeaches - 1, numbeaches*numbeaches))
    for r in range(numbeaches - 1):
        matrixform = num.zeros((numbeaches,numbeaches))
        for s in range(numbeaches):
            matrixform[s,r+1] = -1
```

```
            matrixform[r+1] = num.ones(numbeaches)
            beachmatrix5[r] = num.array([item for innerlist in matrixform for item in
innerlist ])
        beachmatrix5 = num.concatenate((num.zeros([numbeaches - 1, numcargo +
numships+ numbeaches]), beachmatrix5), axis=1)

        #choose locations to visit
        chooselocation2 = num.concatenate((num.zeros([numbeaches, numcargo+ numships
]), beachidentity ), axis =1)
        chooselocation = num.concatenate((chooselocation2, -beachmatrix1), axis =
1)

        cargoinfo = num.vstack((cargowts2, cargoarea2, selectvessel))
        a = num.vstack((cargoinfo, cargoinput, demandinput, beachmatrix, beachma-
trix2, startmatrix, chooselocation, beachmatrix3, beachmatrix4, beachmatrix5))
        cargooutput = [self.lift, self.area, 1]
        b = num.concatenate((cargooutput, num.zeros(numcargo), num.zeros(numcargo),
num.ones(2*numbeaches+2), num.zeros(2*numbeaches), num.ones(((numbeaches - 2)*(num-
beaches - 1))/2), num.zeros(numbeaches-1)),axis=0)

        totalnumvar = numcargo + numships + numbeaches + numbeaches * num-
beaches
        vlb = num.zeros(totalnumvar)
        vub = 10000000000*num.ones(totalnumvar)

        allvariables = num.zeros(totalnumvar)
        for i in range(totalnumvar):
```

```
        allvariables[i] = i + 1
    xint = allvariables
    e = num.concatenate(([-1,-1,0], -1*num.ones(numcargo), -1*num.ones(numcargo),
-1*num.ones(2*numbeaches), num.zeros(2), num.zeros(2*numbeaches), -1*num.ones(((numbeaches
- 2)*(numbeaches - 1))/2), num.zeros(numbeaches-1)), axis=0)

    [obj, x, duals] = lpsolve(f, a, b, e, vlb, vub, xint)
    cargowant1=num.array(x)
    cargowant=cargowant1[0:numcargo]
    loaditem = cargowant1[numcargo:numcargo+numships]
    beachtovisit = cargowant1[numcargo+numships:numcargo+numships+numbeaches]
    beachorder = cargowant1[numcargo+numships+numbeaches:]

    loadoutwt = num.sum(cargowts * cargowant)
    loadoutarea = num.sum(cargoarea * cargowant)
    totaltriptime = num.sum(beachcosts2 * beachorder)
    beachorder = num.reshape(beachorder, (numbeaches,-1))
    beachorder = num.argmax(beachorder, axis=1)
    beachorder2 = num.zeros(num.size(beachorder))
    beachnames = []
    for i in range(numbeaches):
        p = beachorder2[i-1]
        if i >=1 and p == 0:
            beachorder2[i] = 0
        else:
            beachorder2[i] = beachorder[p]
            beaches2 = ['none'] + beaches
```

```
                beachnames.append(beaches2[num.int(beachorder[p])])
        while beachnames[-1] == 'none':
            beachnames.pop()

        relativepriority = num.sum(cargovalue * cargowant)

        if num.max(cargowant) == 0: #if no cargo is selected, set to infinate so the connector does not move forward
            cargowant = num.inf * num.ones(numcargo)
        return cargowant, loadoutarea, loadoutwt, relativepriority, beachorder2, beachnames, totaltriptime

    def checkspotcargo(self, objectsatlocation):
        """Find which given cargo objects (at location) have desired cargo and an available interface.

        Returns an array with entries of one or zero for each object:
        -one signifies a cargo object at the given location with desired cargo and an available interface.
        -zero signifies not a cargo object, not at the given location, or doesn't have desired cargo and an available interface.
        """
        #Identify objects with cargo desired
        if self.purpose == "pull":
            cargotest = num.greaterequal(Cargohave, self.cargowant)
        else:
            cargotest = num.greaterequal(Cargowant, self.cargohave)
```

```
        objectswithcargo = [num.prod(row) for row in cargotest]

        #Identify spots at location with an available interace
        compspots = num.array([self.spots for k in range(Source.assets)])
        availablespots = num.minimum(Spots, compspots)
        spotsatlocation = num.minimum(num.transpose(availablespots), objectsatlo-
cation)

        spotswithcargo = num.transpose(num.minimum(spotsatlocation, objectswith-
cargo))
        return spotswithcargo

    def findspot(self, spotcargotest, j):
        """Find the spot at one of the given cargo objects with the shortest loading
time.

        Returns:
        -shortestrow: The cargo object with the best spot
        -shortestcol: The interface of the best spot
        -shortestwaittime: The wait time for this spot
        -incdiff: An array with entries of zero or one for each interface.
                -one signifies an incompatible interface
                -zero signifies a compatible interface
        """
        #Identify the spot with the shortest waittime
        waits = spotcargotest*self.wait
```

```
        waits[waits <= 0] = num.inf # zeros are replaced with infinity before attempt-
ing to find the minimum waiting time
        shortestrow, shortestcol = num.unravelindex(waits.argmin(), waits.shape)
        shortestwaittime = waits[shortestrow, shortestcol]

        ##Remove the spot being used and apply compatibility
        Taken = num.zeros(numinterfaces)
        Taken[shortestcol] = 1
        Incspots = Incompatible * Taken
        incrow = Incspots.sum(1)
        removinginc = num.maximum(num.zeros(num.shape(incrow)), Spots[shortestrow]
- incrow)
        removinginc2 = num.minimum(removinginc, Spots[shortestrow])
        incdiff = Spots[shortestrow] - removinginc2
        Spots[shortestrow] = removinginc
        Spots[shortestrow, shortestcol] = -1
        ##Remove or add cargo to selected location
        if self.purpose == "pull":
            Cargohave[shortestrow] = Cargohave[shortestrow]-num.array(self.cargowant)
            Cargowant[shortestrow] = Cargowant[shortestrow] + num.array(self.cargowant)
            self.cargohave = self.cargowant
            self.cargowant = num.zeros(numcargo)
            replacematrix(self, j)

        return shortestrow, shortestcol, shortestwaittime, incdiff

    def getcargohave(self, location):
```

"""Identify the cargo items that are at that location:

Returns a 2D array with objects down the rows and cargo items across the columns.

-Values signify an amound of a specific cargo item on an object

-A row of zeros signifies an object that is not at location or has no cargo

"""

cargoatlocation = calcloc(self, location)

locationcargohave = num.transpose([col * cargoatlocation for col in num.transpose(Cargohave

return locationcargohave

def replacematrix(self, j):

"""Track overall cargo delivered and update matrices, demand."""

Goal[j] = self.goal

Location[j] = self.location

Cargohave[j] = self.cargohave

Cargowant[j] = self.cargowant

for beach in beaches:

vars()[beach+'cargohave'] = getcargohave(self, beach)

globals()[beach + 'CargoHave'] = num.sum(eval(beach+'cargohave'), 0)

if numPort > 0: #Update port demand schedule based on cargo delivered and cargo in route

Testgoaltype = calcgoal(self, "beachMEC") + calcgoal(self, "beachLCAC") + calcgoal(self, "beachLCACR") + calcgoal(self, "beachhelo")

for p in range(numPort):

```
            Testgoaltype = Testgoaltype + calcgoal(self, "Port%02d"%(p, ))
        y, x = num.shape(Cargowant)
        Testgoal2 = num.transpose([Testgoaltype for n in range(x)])
        cargoonconnectors = Cargohave*Testgoal2

        Testloctype = num.zeros(len(Names))
        for p in range(numPort):
            Testloctype = Testloctype + calcloc(self, "Port%02d"%(p, ))
        y, x = num.shape(Cargowant)
        Testloc2 = num.transpose([Testloctype for n in range(x)])
        cargoatport = Cargohave*Testloc2

        global Portcargowant
        portcargowant = - cargoonconnectors - cargoatport
        beachschedule = num.zeros(numcargo)
        for beach in beaches:
            beachschedule+= globals()[beach + 'schedule']
        Portcargowant = beachschedule + num.sum(portcargowant, 0)
        Portcargowant2 = beachschedule - num.sum(cargoatport, 0)

        for p in range(numPort):
            Cargowant[Names.index('Port%02d'%(p, ))] = Portcargowant2

def replacematrixcargoship(self, j):
    """Update matrices when cargo ship properties change"""
    Location[j] = self.location
    Type[j] = self.type
```

```
class CargoGenerator(Process):
    """Cargo Generator

        Creates the cargo demand schedule based on the inputted demand.
        Initial demand is set immediately, but additional demand is not generated
        until the set time after the first arrival of a connector on the beach.
        """
    def init(self):
        Process.init(self)
        global CargoNeededOverall
        Daynow = 0
        for beach in beaches:
            Cargowantedall = eval(beach + 'Needed')
            Cargowanted = Cargowantedall[Daynow]
            Cargowanted[11] = Palletdemand
            Cargowanted[24] = Petrodemand
            globals()[beach + 'schedule'] = Cargowanted
            CargoNeededOverall = num.sum(Cargowanted*cargowts)

    def run(self):
        """The PEM for generating cargo.

        Following the first arrival of a connector to the beach, additional demand
        is generated every 24 hours for each day in the input file.
        """
        global CargoNeededOverall
```

```
            yield waituntil, self, self.firstarrival
            Daynow = 0
            yield hold, self, 24*60
            while True:
                Daynow += 1
                if Daynow<(Days):
                    for beach in beaches:
                        Cargowantedall = eval(beach + 'Needed')
                        Cargowanted = Cargowantedall[Daynow]
                        Cargowanted[11] = Palletdemand
                        Cargowanted[24] = Petrodemand
                        globals()[beach + 'schedule'] = globals()[beach + 'schedule'] +
Cargowanted
                        CargoNeededOverall += num.sum(Cargowanted*cargowts)

                        #reactivate(myruntest)
                        checkvessel.signal()
                        yield hold, self, 0
                else:
                    for beach in beaches:
                        Cargowanted = num.zeros(numcargo)
                        Cargowanted[11] = Palletdemand
                        Cargowanted[24] = Petrodemand
                        globals()[beach + 'schedule'] = globals()[beach + 'schedule'] +
Cargowanted
                        CargoNeededOverall += num.sum(Cargowanted*cargowts)
```

```
                #reactivate(myruntest)
                checkvessel.signal()
                yield hold, self, 0
            yield hold, self, demandinterval #interval between demand periods

    def firstarrival(self):
        """Triggers cargo demand based on the arrival of the first connector."""
        MEC.numunloaded = sum([mec.numunloaded for mec in MEC.roster])
        LCAC.numunloaded = sum([lcac.numunloaded for lcac in LCAC.roster])
        LCACR.numunloaded = sum([lcacr.numunloaded for lcacr in LCACR.roster])
        CH46.numunloaded = sum([ch46.numunloaded for ch46 in CH46.roster])
        CH53.numunloaded = sum([ch53.numunloaded for ch53 in CH53.roster])

        arrivals = MEC.numunloaded + LCAC.numunloaded + LCACR.numunloaded
+ CH46.numunloaded + CH53.numunloaded

        if arrivals >= 1:
            return True
        else:
            return False

    def checkallspots(self, objectsatlocation, purpose):
        """Find which given cargo objects (at location) have predicted cargo desired
and usable interface.

        Returns an array with entries of one or zero for each object:
        -one signifies a cargo object at the given location with cargo desired and usable
```

interface

-zero signifies not a cargo object, not at the given location, or doesn't have cargo desired and usable interface.

"""

global Predictedcargo, allSpots

Cargohavepredicted = Cargohave - Predictedcargo

#Identify objects with predicted cargo desired

if purpose == "pull":

cargotest = num.greaterequal(Cargohavepredicted, self.cargowant)

else:

""" At this point in the simulation, the connector is empty so self.cargohave is zero.

self.cargowant is used as a prediction for the future cargo that the connector will have. """

cargotest = num.greaterequal(Cargowant, self.cargowant)

objectswithcargo = [num.prod(row) for row in cargotest]

#Identify objects at location with a usable (not necessarily available) interace

usablespots = num.minimum(allSpots, self.spots)

spotsatlocation = num.minimum(num.transpose(usablespots), objectsatlocation)

spotswithcargo = num.transpose(num.minimum(spotsatlocation, objectswithcargo))

return spotswithcargo

def choosegoal(self):

"""Select a pair of push-pull locations with the minimum travel, queueing, and loading times.

A double nested for loop is used to cycle through every possible push-pull pair.

1. Select push location.

2. Filter pull locations from all possible based on current location and push location.

3. Select cargo to load based on push demand and cargo available at pull point/ vessel

4. Select the push and pull spots with the shortest combined loading and queueing time.

5. Store corresponding data: interface, cargo object, incompatibility vector, loading times vector.

6. Add up queue, loading, and travel times. Record time and data if less than minimum time.

7. Repeat steps 3-6 for all pull locations

8. Repeat steps 1-7 for all push locations

9. Record push and pull data in global predicted arrays (Wait, Cargo, Schedule)

10. Record pull location and push location for connector object

Returns true if a pair was successfully found.

"""

global Predictedwaits

global Predictedcargo

global PredictedSBschedule

global PredictedPortcargowant

#global PredictedBeachschedule

besttime = num.inf

pushlocs = list(self.pushlocs)

currentloc = self.location

currentlocstripped = currentloc.strip("0123456789")

pushlocs2 = []

if self.hopping == 1:

pulllocs = [loc for loc in self.pulllocs if loc.strip("0123456789") not in [currentlocstripped]]

for pullloc in pulllocs:

#Identify pull spot with the shortest combined loading and queue time

cargoatlocation = calcloc(self, pullloc)

self.cargowant = num.zeros(numcargo)

pullcompatiblespots = checkallspots(self, cargoatlocation, "pull")

loadingtimes = pullcompatiblespots * self.wait

queueingtimes = pullcompatiblespots * Predictedwaits

waits = loadingtimes + queueingtimes

waits[waits <= 0] = num.inf # zeros are replaced with infinity before attempting to find the minimum waiting time

shortestrow, shortestcol = num.unravelindex(waits.argmin(), waits.shape)

pullloadingtime = waits[shortestrow, shortestcol]

pullwaittime = queueingtimes[shortestrow, shortestcol]

#apply compatibility

Taken = num.zeros(numinterfaces)

Taken[shortestcol] = 1

Incspots = Incompatible*Taken

```
                incrow = Incspots.sum(1)

                #store pull data
                pulldata = shortestcol, shortestrow, incrow, loadingtimes

                cargodesired = num.zeros((numcargo))
                for beach in beaches:
                    tempschedule, Priority = idpredictedschedule(self,beach)
                    cargodesired = num.vstack((cargodesired,tempschedule))
                    globals()['cargowant' + beach + self.name] = num.zeros(numcargo)
                cargocheck = num.sum(num.sum(cargodesired)) + pullloadingtime +
pullwaittime
                if cargocheck > 0 and cargocheck < num.inf: #Speed up by not running
LP if no cargo is demanded
                    self.cargowant, loadoutarea, loadoutwt, relativepriority, beachorder,
beachnames, totaltriptime = choosehops(self, pullloc, True)
                    Timetopullload = num.maximum( Distances[self.location][pullloc]/
self.speed * 60, pullwaittime) + pullloadingtime
                    Timetopullload = Timetopullload + Distances[pullloc][beachnames[0]]/
self.speed * 60
                    totaltriptime = totaltriptime + Timetopullload
                else:
                    self.cargowant = num.inf * num.ones(numcargo)
                    loadoutarea, loadoutwt, relativepriority, beachorder, beachnames,
totaltriptime = 0, 0, 1, [], [], num.inf

                weightedtotaltime = totaltriptime/ relativepriority
```

```
                #compare to shortest recorded total time
                if weightedtotaltime < besttime:
                    besttime = weightedtotaltime
                    bestpushloc = beachorder
                    bestpushnames = beachnames
                    bestpullloc = pullloc
                    pullcargowant = self.cargowant
                    pullinterface, pullcargoobject, pullinc, pullloadingtimes = pulldata
                    for beach in beaches:
                        globals()['pushinterface'+beach], globals()['pushcargoobject'+beach],
globals()['pushinc'+beach], globals()['pushloadingtimes'+beach] = globals()['shortestcol'+beach],
globals()['shortestrow'+beach], globals()['incrow'+beach], globals()['loadingtimes'+beach]

            for pushtype in pushlocs:
                if pushtype in beaches:
                    continue
                else:
                    pushlocs2.append(pushtype)
        else:
            pushlocs2 = pushlocs

        for pushloc in pushlocs2:
            pushlocstripped = pushloc.strip("0123456789")
            [schedule, priority] = idpredictedschedule(self, pushloc) #use a different
cargo schedule for each push location
            #limit pull locations from being at current location or at push location
```

```
            pulllocs = [loc for loc in self.pulllocs if loc.strip("0123456789") not in
[pushlocstripped, currentlocstripped]]
            if pushlocstripped in ["ISB", "SB"]:
                pulllocs = [loc for loc in self.pulllocs if loc.strip("0123456789") not in
["Port"]]
            for pullloc in pulllocs:
                if self.name.startswith("CH"):
                    self.nextpush = pushloc
                    [wait, loadoutarea, loadoutwt, self.cargowant, relativepriority] = He-
licopter.choosehelocargo(self, True)
                else:
                    self.cargowant, loadoutarea, loadoutwt, relativepriority = choose-
cargo(self, schedule, priority, pullloc)
                if self.range < Distances[pullloc][pushloc] or self.range <
Distances[self.location][pullloc]:
                    self.cargowant = num.inf*num.ones(numcargo)



                #Identify push spot with the shortest combined loading and queue time
                cargoatlocation = calcloc(self, pushloc)
                pushcompatiblespots = checkallspots(self, cargoatlocation, "push")
                loadingtimes = pushcompatiblespots * self.wait
                queueingtimes = pushcompatiblespots * Predictedwaits
                waits = loadingtimes + queueingtimes
                waits[waits <= 0] = num.inf # zeros are replaced with infinity before
attempting to find the minimum waiting time
                shortestrow, shortestcol = num.unravelindex(waits.argmin(), waits.shape)
```

pushloadingtime = waits[shortestrow, shortestcol]

pushwaittime = queueingtimes[shortestrow, shortestcol]

#apply compatibility

Taken = num.zeros(numinterfaces)

Taken[shortestcol] = 1

Incspots = Incompatible * Taken

incrow = Incspots.sum(1)

#store push data

pushdata = shortestcol, shortestrow, incrow, loadingtimes

#Identify pull spot with the shortest combined loading and queue time

cargoatlocation = calcloc(self, pullloc)

pullcompatiblespots = checkallspots(self, cargoatlocation, "pull")

loadingtimes = pullcompatiblespots * self.wait

queueingtimes = pullcompatiblespots * Predictedwaits

waits = loadingtimes + queueingtimes

waits[waits <= 0] = num.inf # zeros are replaced with infinity before attempting to find the minimum waiting time

shortestrow, shortestcol = num.unravelindex(waits.argmin(), waits.shape)

pullloadingtime = waits[shortestrow, shortestcol]

pullwaittime = queueingtimes[shortestrow, shortestcol]

#apply compatibility

Taken = num.zeros(numinterfaces)

```
Taken[shortestcol] = 1
Incspots = Incompatible*Taken
incrow = Incspots.sum(1)

#store pull data
pulldata = shortestcol, shortestrow, incrow, loadingtimes

#calculate total time (travel + queue + loading)
pulldistance = Distances[currentloc][pullloc]
pushdistance = Distances[pullloc][pushloc]
timetopull = pulldistance/self.speed*60
timetopull += max(pullwaittime-timetopull, 0)
timetopush = timetopull + pullloadingtime + pushdistance/self.speed*60
timetopush += max(pushwaittime-timetopush, 0)
alttotaltime = timetopush + pushloadingtime
if(pushlocstripped == "Port" and isinstance(self, MEC)):
    alttotaltime = alttotaltime + 2*self.wait[portpierspot] #add penalty
for Port
weightedtotaltime = alttotaltime/ relativepriority

#compare to shortest recorded total time
if weightedtotaltime < besttime:
    besttime = weightedtotaltime
    bestpushloc = pushloc
    bestpullloc = pullloc
    pullcargowant = self.cargowant
    pullinterface, pullcargoobject, pullinc, pullloadingtimes = pulldata
```

```
                pushinterface, pushcargoobject, pushinc, pushloadingtimes = push-
data
        if besttime == num.nan or besttime == num.inf:  #no push-pull location
found
            return False

        #create new wait time array at pull location along with incompatibility con-
siderations
        self.pullwaittime = num.zeros(Predictedwaits.shape)
        self.pullwaittime[pullcargoobject, pullinterface] = pullloadingtimes[pullcargoobject,
pullinterface]
        self.pullwaittime[pullcargoobject] += (pullinc*pullloadingtimes[pullcargoobject,
pullinterface])

        #create new wait time array at push location along with incompatibility con-
siderations
        self.pushwaittime = num.zeros(Predictedwaits.shape)
        bestpushloc2 = num.array(bestpushloc)
        if num.shape(bestpushloc2)!=():
            for beach in bestpushnames:
                self.pushwaittime[globals()['pushcargoobject'+beach], globals()['pushinterface'+beach
= globals()['pushloadingtimes'+beach] [globals()['pushcargoobject'+beach], globals()['pushinterface
                self.pushwaittime[globals()['pushcargoobject'+beach]] +=
(globals()['pushinc'+beach]*globals()['pushloadingtimes'+beach] [globals()['pushcargoobject'+beac
globals()['pushinterface'+beach]])
        else:
            self.pushwaittime[pushcargoobject, pushinterface] = pushloadingtimes[pushcargoobject,
```

```
pushinterface]
            self.pushwaittime[pushcargoobject] += (pushinc*pushloadingtimes[pushcargoobject,
pushinterface])
        #create predicted cargo use array and add to Predictedcargo
        self.predictedcargo = num.zeros(Predictedcargo.shape)
        self.predictedcargo[pullcargoobject] = pullcargowant
        Predictedcargo = Predictedcargo + self.predictedcargo

        #add to global array
        Predictedwaits = Predictedwaits + self.pullwaittime + self.pushwaittime

        #add predicted cargo to the predicted schedule where the connector is going
to push
        if num.shape(bestpushloc2)!=():
            cargowantcalc = pullcargowant
            for beach in bestpushnames:
                globals()['cargowant' + beach + self.name] = num.minimum(cargowantcalc,
globals()[beach + 'schedule'])
                cargowantcalc = cargowantcalc - globals()['cargowant' + beach + self.name]
                globals()['Predicted'+ beach +'schedule'] += globals()['cargowant' +
beach + self.name]

            bestpushloc = bestpushnames
            self.nexthop = beachnames

        else:
            for beach in beaches:
```

```
                if beach == bestpushloc:
                    globals()['cargowant' + beach + self.name] = pullcargowant
                else:
                    globals()['cargowant' + beach + self.name] = num.zeros(numcargo)

            locstripped = bestpushloc.strip("0123456789")
            if(locstripped == "SB"):
                PredictedSBschedule += pullcargowant
            elif(locstripped == "Port"):
                PredictedPortcargowant += pullcargowant
            else:
                globals()['Predicted'+ bestpushloc +'schedule'] += pullcargowant

        self.goal = bestpullloc
        self.nextpush = bestpushloc
        self.cargowant = pullcargowant
        return True

    def idschedule(self,location):
        """Identify which cargo schedule to use based on given location."""
        locstripped = location.strip("0123456789")

        if(locstripped == "SB"):
            schedule = SBschedule
            priority = SBcargopriority
        elif(locstripped == "Port"):
            schedule = Portcargoneeds(self)
```

```
            priority = Portcargopriority
        else:
            schedule = globals()[location + 'schedule']
            priority = Beachcargopriority

        return [schedule, priority]

    def idpredictedschedule(self,location):
        """Identify which predicted cargo schedule to use based on given location."""
        locstripped = location.strip("0123456789")
        if(locstripped == "SB"):
            schedule = SBschedule - PredictedSBschedule
            priority = SBcargopriority
        elif(locstripped == "Port"):
            schedule = Portcargoneeds(self) - PredictedPortcargowant
            priority = Portcargopriority
        else:
            schedule = globals()[location + 'schedule'] - globals()['Predicted' + location +'schedule']
            priority = Beachcargopriority

        schedule = [max(s, 0) for s in schedule]
        return [schedule, priority]

    def updateschedule(self):
        """Update cargo schedule based on push location."""
        global SBschedule
```

```
        global Portcargowant
        #global Beachschedule
        locstripped = self.nextpush.strip("0123456789")
        if(locstripped == "SB"):
            SBschedule = SBschedule - self.cargohave
        elif(locstripped == "Port"):
            Portcargowant = Portcargowant - self.cargohave
        else:
            if self.hopping == 1:
                for beach in beaches:
                    globals()[beach + 'schedule'] = globals()[beach + 'schedule'] - glob-
als()['cargowant' + beach + self.name]
            else:
                globals()[self.nextpush + 'schedule'] = globals()[self.nextpush + 'sched-
ule'] - self.cargohave
            priority = Beachcargopriority

        if self.goal.startswith("SB"): #add demand to SB schedule if pulled from SB
            SBschedule = SBschedule + self.cargohave

    def lpsolve(f = None, a = None, b = None, e = None, vlb = None, vub = None,
xint = None, scalemode = None, keep = None):
    """LPSOLVE Solves mixed integer linear programming problems.
        Open source PEM available at: http://lpsolve.sourceforge.net/5.5/Python.htm
    SYNOPSIS: [obj, x, duals, stat] = lpsolve(f, a, b, e, vlb, vub, xint, scalemode,
keep)
        solves the MILP problem
```

```
        max v = f'*x
        a*x <> b
            vlb <= x <= vub
            x(int) are integer
ARGUMENTS: The first four arguments are required:
        f: n vector of coefficients for a linear objective function.
        a: m by n matrix representing linear constraints.
        b: m vector of right sides for the inequality constraints.
        e: m vector that determines the sense of the inequalities:
                e(i) = -1 ==> Less Than
                e(i) = 0 ==> Equals
                e(i) = 1 ==> Greater Than
        vlb: n vector of lower bounds. If empty or omitted,
            then the lower bounds are set to zero.
        vub: n vector of upper bounds. May be omitted or empty.
        xint: vector of integer variables. May be omitted or empty.
scalemode: scale flag. Off when 0 or omitted.
        keep: Flag for keeping the lp problem after it's been solved.
            If omitted, the lp will be deleted when solved.
OUTPUT: A nonempty output is returned if a solution is found:
        obj: Optimal value of the objective function.
        x: Optimal value of the decision variables.
    duals: solution of the dual problem."""

if f == None:
    help(lpsolve)
    return
```

```
m = len(a)
n = len(a[0])
lp = lpsolve('makelp', m, n)
lpsolve('setverbose', lp, IMPORTANT)
lpsolve('setmat', lp, a)
lpsolve('setrhvec', lp, b)
lpsolve('setobjfn', lp, f)
lpsolve('setmaxim', lp) # default is solving minimum lp.

for i in range(m):
    if e[i] < 0:
        contype = LE
    elif e[i] == 0:
        contype = EQ
    else:
        contype = GE
    lpsolve('setconstrtype', lp, i + 1, contype)

if vlb != None:
    for i in range(n):
    lpsolve('setlowbo', lp, i + 1, vlb[i])

if vub != None:
    for i in range(n):
    lpsolve('setupbo', lp, i + 1, vub[i])
```

```
if xint != None:
    for i in range(len(xint)):
    lpsolve('setint', lp, xint[i], 1)

if scalemode != None:
    if scalemode != 0:
    lpsolve('setscaling', lp, scalemode)

result = lpsolve('solve', lp)
if result == 0 or result == 1 or result == 11 or result == 12:
    [obj, x, duals, ret] = lpsolve('getsolution', lp)
    stat = result
else:
    obj = []
    x = []
    duals = []
    stat = result

if keep == None and keep != 0:
    lpsolve('deletelp', lp)

return [obj, x, duals]

## Model ————————————————
Source.assets = 0
MEC.loadwts = []
MEC.loadareas = []
```

```
    LCAC.loadwts = []
    LCAC.loadareas = []
    LCACR.loadwts = []
    LCACR.loadareas = []
    MLP.SBqueue = 0
    MLP.beachqueue = 0
    MLP.travel = 0
    MLP.fuel = 0
    CH46.loadwts = []
    CH46.loadareas = []
    CH53.loadwts = []
    CH53.loadareas = []

    initialize()
    waiting = []
    #myruntest = RunTest("my run test")
    #activate(myruntest, myruntest.run())
    beachemptyLCAC = []
    beachemptyLCACR = []
    SBLCAC = []
    SBLCACR = []
    for beach in beaches:
        globals()[beach+'emptyLCAC'] = Store(capacity = 1000, initialBuffered =
waiting)
        beachemptyLCAC = beachemptyLCAC + [beach+'emptyLCAC']
        globals()[beach+'SBLCAC'] = Store(capacity = 1000, initialBuffered = wait-
ing)
```

```
        SBLCAC = SBLCAC + [beach+'SBLCAC']
        globals()[beach+'emptyLCACR'] = Store(capacity = 1000, initialBuffered =
waiting)
        beachemptyLCACR = beachemptyLCACR + [beach+'emptyLCACR']
        globals()[beach+'SBLCACR'] = Store(capacity = 1000, initialBuffered = wait-
ing)
        SBLCACR = SBLCACR + [beach+'SBLCACR']
    ISBLCAC = Store(capacity = 1000, initialBuffered = waiting)
    ISBLCACR = Store(capacity = 1000, initialBuffered = waiting)
    ISBCH46 = Store(capacity = 1000, initialBuffered = waiting)
    ISBCH53 = Store(capacity = 1000, initialBuffered = waiting)
    ISBINLS = Store(capacity = 1000, initialBuffered = waiting)
    ISBLCU1600 = Store(capacity = 1000, initialBuffered = waiting)
    MLPload = Resource(capacity = 1, name = "MLP",unitName = "Spot")
    s = Source()
    activate(s, s.generate(number1 = NumMEC, number2 = NumMLP, number3 =
NumLMSR, number5 = NumLCAC, number7 = numJHSV, number8 = numISB,
number9 = NumLCACR, number12 = NumLHD, number13 = numMV22, num-
ber14 = numCH46, number15 = numCH53, number18 = numLCU2000, number19
= numLSV, number20 = numPort, number21 = numTAKE, number22 = numINLS,
number24 = NumLPD, number25 = NumLSD, number26 = NumLCU1600, MLPload-
control = MLPload),at = 0.0)
    simulate(until = maxTime)
outputfile.close()
```

REFERENCES

[1] "Global security." www.globalsecurity.org.

[2] "Interstate products, inc." www.interstateproducts.com.

[3] "Military sealift command 2000 in review." www.msc.navy.mil/annualreport/2000/exercise.htm.

[4] "Simpy simulation package homepage." simpy.sourceforge.net.

[5] "Joint pub 3-07 joint doctrine for military operations other than war." Joint Chiefs of Staff, June 16 1995.

[6] "Making peace while staying ready for war: The challenges of u.s. military participation in peace operations." Congressional Budget Office, December 1999.

[7] "Marine corps order 3120.9b, policy for marine expeditionary unit (special operations capable (meu (soc))," September 2001.

[8] "Navy test vessel may yield new warfare concepts," *Pentagon Brief*, pp. 8–8, Jan 15 2002.

[9] "Defense science board task force on seabasing." Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, Aug 2003.

[10] "Transformation planning guidance," April 2003.

[11] "Civil-military relationship in complex emergencies - an iasc reference paper." Inter-Agency Standing Committee, June 2004.

[12] "The role of experimentation in building future naval forces." Committee for the Role of Experimentation in Building Future Naval Forces, 2004.

[13] "Joint publication 4-01.6: Joint logistics over-the-shore (jlots)." Joint Chiefs of Staff, August 5 2005.

[14] "Onr baa announcement 05-020, sea base connector transformable-craft (t-craft) prototype demonstrator." Office of Naval Research, 2005.

[15] "Sea basing: Ensuring joint force access from the sea." Committee on Sea Basing: Ensuring Joint Force Access from the Sea, 2005.

[16] "Sea basing joint integrating concept, version 1.0." Department of Defense, August 2005.

[17] "Unified course 2005 seabasing analysis final report." Navy Warfare Development Command, 2006.

[18] "Seabasing of the range of military operations." United States Marine Corps, March 26 2009.

[19] "Defense acquisition guidebook." dag.dau.mil, February 19 2010.

[20] "The first quadrennial diplomacy and development review: Leading through civilian power." U.S. Department of State, December 2010.

[21] "Military sealift command ships.," *Sea Power*, vol. 53, no. 1, p. 42, 2010.

[22] "Military sealift command completes annual resupply mission to antarctica." Military Sealift Command Public Affairs, February 14 2011.

[23] Afonso, P. M. and da Conceição Cunha, M., "Robust optimal design of activated sludge bioreactors," *Journal of Environmental Engineering*, vol. 133, no. 1, pp. 44–52, 2007.

[24] Agostini, H., "1st marine expeditionary brigade departs for dawn blitz," tech. rep., Federal Information & News Dispatch, Inc, Sep 28 2011.

[25] Annati, M., "Naval support for combat and supply.," *Military Technology*, vol. 33, no. 7, pp. 73 – 81, 2009.

[26] Anonymous, "Military sealift command," *Defense Transportation Journal*, vol. 55, no. 2, pp. 30–30–31, 1999.

[27] Anonymous, "Supplying the operating forces," *Newsletter - United States.Navy Supply Corps*, vol. 65, no. 5, pp. 2–2, 2002.

[28] Antony, J., Anand, R. B., Kumar, M., and Tiwari, M., "Multiple response optimization using taguchi methodology and neuro-fuzzy based model," *Journal of Manufacturing Technology Management*, vol. 17, no. 7, pp. 908–925, 2006.

[29] Arellano-Garcia, H. and Wozny, G., "Chance constrained optimization of process systems under uncertainty: I. strict monotonicity," *Computers and Chemical Engineering*, vol. 33, no. 10, pp. 1568 – 1583, 2009. Selected Papers from the 18th European Symposium on Computer Aided Process Engineering (ESCAPE-18).

[30] Armony, M., "Dynamic routing in large-scale service systems with heterogeneous servers," *Queueing Systems*, vol. 51, no. 3-4, pp. 287–287–329, 2005.

[31] Arvidsson, M. and Gremyr, I., "Principles of robust design methodology.," *Quality & Reliability Engineering International*, vol. 24, no. 1, pp. 23 – 35, 2008.

[32] Auerbach, P. S., Norris, R. L., Menon, A. S., Brown, I. P., and et al., "Civil-military collaboration in the initial medical response to the earthquake in haiti," *The New England Journal of Medicine*, vol. 362, p. e32, Mar 11 2010.

[33] Badiru, A. B. and Ayeni, B. J., *Practitioner's Guide to Quality and Process Improvement*. Chapman and Hall, 1993.

[34] Bahouth, A., Crites, S., Matloff, N., and Williamson, T., "Revisiting the issue of performance enhancement of discrete event simulation software," in *Simulation Symposium, 2007. ANSS '07. 40th Annual*, pp. 114 –122, march 2007.

[35] Bastian, C. and Kan, A. H. G. R., "The stochastic vehicle routing problem revisited," *European Journal of Operational Research*, vol. 56, no. 3, pp. 407 – 412, 1992.

[36] Bates, R. A., Wynn, H. P., and Fraga, E. S., "Feasible region approximation: a comparison of search cone and convex hull methods.," *Engineering Optimization*, vol. 39, no. 5, pp. 513 – 527, 2007.

[37] Berkelaar, M., Eikland, K., and Notebaert, P., "lp solve reference guide." http://lpsolve.sourceforge.net/5.5/, April 2011.

[38] Bertsimas, D. and Chryssikou, T., "Bounds and policies for dynamic routing in loss networks," *Operations Research*, vol. 47, no. 3, pp. pp. 379–394, 1999.

[39] Bertsimas, D. J., "A vehicle routing problem with stochastic demand.," *Operations Research*, vol. 40, no. 3, p. 574, 1992.

[40] Bessler, M. and Seki, K., "Civil-military relations in armed conflicts: A humanitarian perspective," *Liaison*, vol. III, no. 3, pp. 4–10, 2006.

[41] Bingham, D., Sitter, R. R., and Tang, B., "Orthogonal and nearly orthogonal designs for computer experiments.," *Biometrika*, vol. 96, no. 1, p. 51, 2009.

[42] Boensel, M. and Schrady, D., "Jelo: A model of joint expeditionary logistics operations." Naval Postgraduate School, October 2004.

[43] Bonabeau, E., "Agent-based modeling: Methods and techniques for simulating human systems," in *Proceedings of the National Academy of Sciences*, vol. 99 (suppl. 3), 2002.

[44] Borshchev, A. and Filippov, A., "From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools," in *The 22nd International Conference of the System Dynamics Society*, July 25-29 2004.

[45] Bradfield, R., Wright, G., Burt, G., Cairns, G., and Heijden, K. V. D., "The origins and evolution of scenario techniques in long range business planning," *Futures*, vol. 37, no. 8, pp. 795 – 812, 2005.

[46] Bradley, S. P., Hax, A. C., and Magnanti, T. L., *Applied Mathematical Programming*. Addison-Wesley, 1977.

[47] Buckingham, R. H., "Navy mh-53es can support the marines," *United States Naval Institute.Proceedings*, vol. 128, no. 7, pp. 76–76–77, 2002.

[48] Buckingham, R. H., "Navy mh-53s can support the marines," *Marine Corps Gazette*, vol. 87, no. 3, pp. 29–29–30, 2003.

[49] Buckwalter, B., "1st marine expeditionary brigade exercises rapid response capability," tech. rep., Federal Information & News Dispatch, Inc, Jul 22 2011.

[50] Buckwalter, B., "What is a marine expeditionary brigade?," tech. rep., Federal Information & News Dispatch, Inc, Jul 25 2011.

[51] Button, R., Blickstein, I., Gordon, J., Wilson, P., and Riposo, J., *A Preliminary Investigation of Ship Acquisition Options for Joint Forcible Entry Operations*. RAND Corporation, 2005.

[52] Button, R. W., IV, J. G., Riposo, J., Blickstein, I., and Wilson, P. A., "Warfighting and logistic support of joint forces from the joint sea base." RAND, 2007.

[53] Calvin, J. M. and Leung, J. Y. T., "Average-case analysis of a greedy algorithm for the 0/1 knapsack problem," *Operations Research Letters*, vol. 31, no. 3, pp. 202 – 210, 2003.

[54] Capkovic, F., "A solution of deds control synthesis problems.," *Systems Analysis Modelling Simulation*, vol. 42, no. 3, pp. 405 – 414, 2002.

[55] Carroll, S. J. and Isaacson, K., "The potential effects of alternative concepts for managing the distribution of resupply cargo." RAND, 1992.

[56] Cason, R. K., *Analysis of the Vertical Rakeoff and Landing Unmanned Aerial Vehicle (VTUAV) in Small UNit Urban Operations*. PhD thesis, Naval Postgraduate School, September 2004.

[57] Castillo, V., "Parallel simulations of manufacturing processing using simpy, a python-based discrete event simulation tool," in *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, p. 2294, dec. 2006.

[58] CDM Technologies, I., "T-craft for humanitarian aid assessment effort annotated brief." July 21 2010.

[59] Chandra, C. and Grabis, J., *Supply Chain Configuration: Concepts, Solutions, and Applications*. Springer, 2007.

[60] Chen, L.-H. and Chen, Y.-H., "A computer-simulation-oriented design procedure for a robust and feasible job shop manufacturing system," *Journal of Manufacturing Systems*, vol. 14, no. 1, pp. 1–1, 1995.

[61] Chen, W., Allen, J. K., Tsui, K.-L., and Mistree, F., "A procedure for robust design: Minimizing variations caused by noise factors and control factors," *Journal of Mechanical Design*, vol. 118, no. 4, pp. 478–485, 1996.

[62] Chen, W., Jin, R., and Sudjianto, A., "Analytical global sensitivity analysis and uncertainty propagation for robust design," *Journal of Quality Technology*, vol. 38, no. 4, pp. 333–348, 2006.

[63] Cioppa, T. M. and Lucas, T. W., "Efficient nearly orthogonal and space-filling latin hypercubes," *Technometrics*, vol. 49, no. 1, pp. 45–55, 2007.

[64] Clark, A. V., "Seapower 21: Projecting decisive joint capabilities," in *Proceedings of the U.S. Naval Institute*, October 2002.

[65] Cooper, K., "Seabasing enabler inp program." Naval S&T Partnership Conference, 2006.

[66] Dano, S., *Nonlinear and Dynamic Programming*. Spring-Verlag, 1975.

[67] Dantzig, G. B. and Ramser, J. H., "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[68] Di Paola, D., Naso, D., Turchiano, B., Cicirelli, G., and Distante, A., "Matrix-based discrete event control for surveillance mobile robotics," *Journal of Intelligent and Robotic Systems*, vol. 56, pp. 513–541, 2009.

[69] Dror, M., "Modeling vehicle routing with uncertain demands as a stochastic program: Properties of the corresponding solution," *European Journal of Operational Research*, vol. 64, no. 3, pp. 432 – 441, 1993.

[70] Dror, M., Laporte, G., and Trudeau, P., "Vehicle routing with stochastic demands: Properties and solution frameworks," *Transportation Science*, vol. 23, no. 3, pp. 166–176, 1989.

[71] Du, L., Wu, J., and Hu, F., "Logistics network design and optimization of closed-loop supply chain based on mixed integer nonlinear programming model," in *ISECS International Colloquium on Computing, Communication, Control, and Management*, 2009.

[72] Elsayed, K. and Elokda, A., "Augmented integrated routing scheme for routing bandwidth-guaranteed connections in internet protocol/multi-protocol label switching over wavelength division multiplexing networks," *Communications, IET*, vol. 5, pp. 1351 –1360, 1 2011.

[73] Erwin, S. I., "Weighed down by heavy hardware, marine go on a diet.," *National Defense*, vol. 95, no. 686, pp. 24 – 25, 2011.

[74] Feillet, D., Dejax, P., and Gendreau, M., "Traveling salesman problems with profits.," *Transportation Science*, vol. 39, no. 2, pp. 188 – 205, 2005.

[75] Feist, T., "Transformation has limits," *United States Naval Institute Proceedings*, vol. 131, no. 4, pp. 65–65–69, 2005.

[76] Fisher, C., "West coast marine expeditionary brigade reestablished," *Leatherneck*, vol. 83, no. 2, pp. 42–42–43, 2000.

[77] Fishman, G. S., *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer, 2001.

[78] Frankis, D., Corrigan, N., and Bailey, R., "Modelling military requirements for non-warfighting operations," in *Simulation Conference Proceedings, 1999 Winter*, vol. 2, pp. 1125 –1130 vol.2, 1999.

[79] Fratarangelo, P., Gale, P., Hancock, W., Katz, D., Kohn, E., Neal, W., Ness, R., Polmar, N., Tozzi, J. T., Webber, G., Weldon, W. F., and Winston, P., "Sea basing," 2005.

[80] Fulghum, D. A., "Pentagon again requests civil helos for resupply," *Aviation Week & Space Technology*, vol. 145, pp. 25–25, Oct 28 1996.

[81] Galloway, C. C., "Seabasing: A transcom perspective," October 2005.

[82] Gao, J., "Optimization of distribution routing problem based on travel time reliability," in *Information Management, Innovation Management and Industrial Engineering, 2009 International Conference on*, vol. 1, pp. 19 –22, dec. 2009.

[83] Gawiejnowicz, S., *Time-Dependent Scheduling*. Springer-Verlag Berlin Heidelberg, 2008.

[84] Gendreau, M., Laporte, G., and Semet, F., "A branch-and-cut algorithm for the undirected selective traveling salesman problem," *Networks*, vol. 32, p. 263273, 1998.

[85] Gibbs, M., "Python wrap-up," *Network World*, vol. 20, pp. 34–34, Jun 16 2003.

[86] Giordano, V., Ballal, P., Lewis, F., Turchiano, B., and Zhang, J. B., "Supervisory control of mobile sensor networks: math formulation, simulation, and implementation," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, pp. 806 –819, Aug 2006.

[87] Giordano, V., Zhang, J., Naso, D., Wong, M., Lewis, F., and Carbotti, A., "Matrix-based discrete event control of automated material handling systems," in *Decision and Control, 2006 45th IEEE Conference on*, pp. 6074 –6079, December 2006.

[88] Giordano, V., Zhang, J. B., Naso, D., and Lewis, F., "Integrated supervisory and operational control of a warehouse with a matrix-based approach," *Automation Science and Engineering, IEEE Transactions on*, vol. 5, pp. 53 –70, January 2008.

[89] Giua, A., *Petri Nets as Discrete Event Models for Supervisory Control*. PhD thesis, Rensselaer Polytechnic Institute, July 1992.

[90] Graybeal, W. J. and Pooch, U. W., *Simulation: Principles and Methods*. Winthrop Publishers, Inc., 1980.

[91] Hagan, R. M., "Modeling sea-based containment of marine expeditionary unit (special operations capable) (meu(soc)) operations ashore," Master's thesis, Naval Postgraduate School, September 1998.

[92] Hamid, U., "Petri net versus queuing theory for evaluation of flexible manufacturing systems," *Advances in Production Engineering and Management*, vol. 5, no. 2, pp. 93–100, 2010.

[93] Hammond, A., "3 global scenarios: Choosing the world we want," *The Futurist*, vol. 33, no. 4, pp. 38–38–43, 1999.

[94] Henderson, K. T., "Army special operations forces and marine expeditionary unit (special operations capable) integration: Something a joint task force commander should consider." School of Advanced Military Studies United States Army Command and General Staff College, May 2004.

[95] House, T. W., "National security strategy," May 2010.

[96] Joint Publication 1-02, *DOD Dictionary of Military and Associated Terms*, 08 November 2010.

[97] Kaku, B. K., Assad, A. A., and Wang, C., *Models for Tool Management in Flexible Manufacturing Systems*, ch. 19, pp. 393–358. Quorum Books, 1998.

[98] Karabuk, S. and Grant, F. H., "A common medium for programming operations-research models," *IEEE Software*, vol. Sept/Oct 2007, pp. 39–47, 2007.

[99] Keeter, H., "Marines to revive expeditionary brigade," *Defense Daily*, vol. 204, pp. 1–1, Nov 02 1999.

[100] Khoshnevis, B., *Discrete Systems Simulation*. McGraw-Hill, Inc., 1994.

[101] Kime, P., "The army's navy," *Sea Power*, vol. 50, no. 4, pp. 33–33–34,36, 2007.

[102] Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., and Cioppa, T. M., "A users guide to the brave new world of designing simulation experiments," *INFORMS Journal on Computing*, vol. 17, no. 3, pp. 263–289, 2005.

[103] Klemmt, A., Horn, S., Weigert, G., and Wolter, K.-J., "Simulation-based optimization vs mathematical programming: A hybrid approach for optimizing scheduling problems," *Robotics and Computer-Integrated Manufacturing*, vol. 25, pp. 917–925, 2009.

[104] Koullias, S., Balestrini-Robinson, S., and Mavris, D., "Surface effect ship sizing and synthesis: A nonlinear programming approach," in *11th International Conference on Fast Sea Transportation*, (Honolulu, Hawaii), Sept 2011.

[105] Krulak, C., "Within striking distance & ready to act.," *U.S. Naval Institute Proceedings*, vol. 125, no. 5, p. 50, 1999.

[106] Laporte, G., "Fifty years of vehicle routing," *Transportation Science*, vol. 43, pp. 408–416, November 2009.

[107] Laporte, G., Louveaux, F. V., and Hamme, L. V., "An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands," *Operations Research*, vol. 50, pp. 415–423, May/June 2002.

[108] Laporte, G., Louveaux, F., and Mercure, H., "Models and exact solutions for a class of stochastic location-routing problems," *European Journal of Operational Research*, vol. 39, no. 1, pp. 71 – 78, 1989.

[109] Larsen, A., *The Dynamic Vehicle Routing Problem*. PhD thesis, Technical University of Denmark, June 2000.

[110] Lasica, K. A., "31st meu earning special operations capability," *Leatherneck*, vol. 82, no. 10, pp. 26–26–27+, 1999.

[111] Lenderink, E. and Stehouwer, P., "Optimization, sensitivity analysis, and robust design using response surface modeling," in *Proc. of SPIE Vol. 7103*, 2008.

[112] Lewis, F., Bogdan, S., Gurel, A., and Pastravanu, O., "Analysis of deadlocks and circular waits using a matrix model for discrete event systems," in *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, vol. 4, pp. 4080 –4085 vol.4, Dec. 1997.

[113] Li, P., Arellano-Garcia, H., and Wozny, G., "Chance constrained programming approach to process optimization under uncertainty," *Computers and Chemical Engineering*, vol. 32, no. 1-2, pp. 25 – 45, 2008.

[114] Li, Y. and Wonham, W., "Control of vector discrete-event systems. i. the base model," *Automatic Control, IEEE Transactions on*, vol. 38, pp. 1214 –1227, aug 1993.

[115] Liao, T.-Y. and Hu, T.-Y., "An object-oriented evaluation framework for dynamic vehicle routing problems under real-time information," *Expert Systems with Applications*, vol. 38, no. 10, pp. 12548 – 12558, 2011.

[116] Lin, C. D., Mukerjee, R., and Tang, B., "Construction of orthogonal and nearly orthogonal latin hypercubes.," *Biometrika*, vol. 96, no. 1, p. 243, 2009.

[117] Lin, W. and Kumar, P., "Optimal control of a queueing system with two heterogeneous servers," *Automatic Control, IEEE Transactions on*, vol. 29, pp. 696 – 703, aug 1984.

[118] Linderman, K. and Choo, A. S., "Robust economic control chart design," *IIE Transactions*, vol. 34, pp. 1069–1078, 2002.

[119] Linderoth, J. T. and Ralphs, T. K., *Integer Programming: Theory and Practice*, ch. Noncommercial Software for Mixed-Integer Linear Programming, pp. 253–303. Taylor & Francis, 2006.

[120] Luh, H. P. and Viniotis, I., "Threshold control policies for heterogeneous server systems," *Mathematical Methods of Operations Research*, vol. 55, pp. 121–142, 2002.

[121] MacCarley, MarkColeman, B. F., "The 8th theater sustainment command leads the way during operation pacific strike 2008.," *Army Logistician*, vol. 41, no. 2, p. 24, 2009.

[122] Mak, T., Cheung, P., Lam, K.-P., and Luk, W., "Adaptive routing in network-on-chips using a dynamic-programming network," *Industrial Electronics, IEEE Transactions on*, vol. 58, pp. 3701 –3716, Aug 2011.

[123] Mak, V. and Thomadsen, T., "Facets for the cardinality constrained quadratic knapsack problem and the quadratic selective travelling salesman problem," tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, nov 2004.

[124] Matloff, N., "Introduction to discrete-event simulation and the simpy language," Feb 2008.

[125] Maxwell, D. T., "An overview of the joint warfare system (jwars)," *Phalanx*, July 2000.

[126] McCarthy, J., ed., *Seabasing Logistics Enabling Concept*. 2000 Navy Pentagon, Washington, DC 20350-2000: Department of the Navy, December 2006.

[127] McLeroy, C., "History of military gaming," *Soldiers*, vol. 63, no. 9, pp. 4–4–6, 2008.

[128] Mireles, J., J. and Lewis, F., "Intelligent material handling: development and implementation of a matrix-based discrete-event controller," *Industrial Electronics, IEEE Transactions on*, vol. 48, pp. 1087 –1097, Dec. 2001.

[129] Montgomery, D. C., "Experimental design for product and process design and development," *Journal of the Royal Statistical Society. Series D (The Statistician)*, vol. 48, no. 2, pp. pp. 159–177, 1999.

[130] Morgan, E. A., "Analysis of high-speed vessels for seventh fleet logistics support," Master's thesis, Naval Postgrtaduate School, 2005.

[131] Motor Transport (PMM 151),, Quantico, VA, *Principal Technical Characteristics Of U.S. Marine Corps Motor Transport Equipment*, tm 11240-od ed., August 2007.

[132] Müller, K. G., "Advanced systems simulation capabilities in simpy."

[133] Mulvey, S. M. and Vanderbei, R. J., "Robust optimization of large-scale systems.," *Operations Research*, vol. 43, no. 2, p. 264, 1995.

[134] Murty, K. G., "Optimization models for decision making: Volume 1." University of Michigan, Ann Arbor, 2003.

[135] Narciso, M., Piera, M. A., and Guasch, A., "A methodology for solving logistic optimization problems through simulation," *SIMULATION*, vol. 86, no. 5-6, pp. 369–389, May/June 2010.

[136] Nauss, R. M., "The elastic generalized assignment problem," *The Journal of the Operational Research Society*, vol. 55, no. 12, pp. pp. 1333–1341, 2004.

[137] Nauss, R. M., *Integer Programming: Theory and Practice*, ch. The Generalized Assignment Problem, pp. 39–55. Taylor & Francis, 2006.

[138] Novoa, C. and Storer, R., "An approximate dynamic programming approach for the vehicle routing problem with stochastic demands," *European Journal of Operational Research*, vol. 196, no. 2, pp. 509 – 515, 2009.

[139] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, *Systems Engineering Guide for Systems of Systems Version 1.0*, washington, dc ed., August 2008.

[140] O'Rourke, R., "Navy lpd-17 amphibious ship procurement: Background, issues, and options for congress." Congressional Research Service, March 16 2011.

[141] Osmundson, J. S., Gottfried, R., Kum, C. Y., Boon, L. H., Lian, L. W., Patrick, P. S. W., and Thye, T. C., "Process modeling: A systems engineering tool for analyzing complex systems," *Systems Engineering*, vol. 7, June 2004.

[142] Padberg, M. and Rinaldi, G., "A branch-and-cut approach to a traveling salesman problem with side constraints," *Management Science*, vol. 35, no. 11, pp. pp. 1393–1412, 1989.

[143] Parson, D., *TLoaDS gen5 Reference Manual*. Simulation Dynamics, Inc., 2001.

[144] Parsons, D. and Krause, L., "Tactical logistics and distribution system (tloads) simulation," in *Simulation Conference Proceedings*, vol. 2, pp. 1174–1178, Dec 1999.

[145] Pavone, M., Frazzoli, E., and Bullo, F., "Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment," *Automatic Control, IEEE Transactions on*, vol. 56, pp. 1259 –1274, june 2011.

[146] Phadke, M. S., *Quality Engineering Using Robust Design*. Prentice Hall, 1989.

[147] Pooch, U. W. and Wall, J. A., *Discrete Event Simulation: A Practical Approach*. CRC Press, 1993.

[148] Potvin, J.-Y., "Evolutionary algorithms for vehicle routing," *Journal on Computing*, vol. 21, pp. 518–548, Fall 2009.

[149] Psaraftis, H., *Vehicle Routing: Methods and Studies*, ch. Dynamic Vehicle Routing Problems, pp. 223–248. Elsevier Science Publishers B.V., 1988.

[150] Psaraftis, H., Orlin, J., Bienstock, D., and Thompson, P., "Analysis and solution algorithms of sealift routing and scheduleing problems: Final report," tech. rep., Sloan School of Management, M.I.T., August 1985.

[151] Psaraftis, H. N., "Dynamic vehicle routing: Status and prospects.," *Annals of Operations Research*, vol. 61, no. 1-4, pp. 143 – 164, 1995.

[152] Rappoport, H. K., Levy, L. S., Golden, B. L., and Toussaint, K. J., "A planning heuristic for military airlift," *Interfaces*, vol. 22, no. 3, pp. pp. 73–87, 1992.

[153] Ren, F., He, T., Das, S. K., and Lin, C., "Traffic-aware dynamic routing to alleviate congestion in wireless sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, pp. 1585 –1599, sept. 2011.

[154] Roorda, M. J., Cavalcante, R., McCabe, S., and Kwan, H., "A conceptual framework for agent-based modelling of logistics services," *Transportation Research Part E*, vol. 46, pp. 18–31, 2010.

[155] Rounsevell, M. D. A. and Metzger, M. J., "Developing qualitative scenario storylines for environmental change assessment," *Wiley Interdisciplinary Reviews: Climate Change*, vol. 1, no. 4, pp. 606–619, 2010.

[156] Rudderham, M. A., "Canada and united nations peace operations," *International Journal*, vol. 63, no. 2, pp. 359–359–384, 2008.

[157] Rykov, V., "Monotone control of queueing systems with heterogeneous servers," *Queueing Systems*, vol. 37, pp. 391–403, 2001.

[158] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., "Design and analysis of computer experiments," *Statistical Science*, vol. 4, no. 4, pp. pp. 409–423, 1989.

[159] Sanchez, S. M., "Nolhdesigns spreadsheet version 4." http://diana.cs.nps.navy.mil/SeedLab/, 2005. accessed 12/21/2011.

[160] Sanchez, S. M., "A robust design tutorial," in *Proceedings of the 1994 Winter Simulation Conference* (Tew, J. D., Manivannan, S., Sadowski, D. A., and Seila, A. F., eds.), 1994.

[161] Sanchez, S. M., "Design of experiments: robust design: seeking the best of all possible worlds," in *WSC '00: Proceedings of the 32nd conference on Winter simulation*, (San Diego, CA, USA), pp. 69–76, Society for Computer Simulation International, 2000.

[162] Sanchez, S. M., Sanchez, P. J., Ramberg, J. S., and Moeeni, F., "Effective engineering design through simulation," *International Transactions on Operational Research*, vol. 3, pp. 169–185, 1996.

[163] Sarkar, S., Yen, H.-H., Dixit, S., and Mukherjee, B., "A novel delay-aware routing algorithm (dara) for a hybrid wireless-optical broadband access network (woban)," *Network, IEEE*, vol. 22, pp. 20 –28, may-june 2008.

[164] Sarri, P., Ait Hssain, A., and Niel, E., "Control of vector discrete-event systems in the context of operational safety," in *Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on*, pp. 263 –268, sep 1997.

[165] Scheibe, S., "Assessment of the operational requirements for the transdormable craft in seabasing mission," Master's thesis, Naval Postgraduate School, June 2010.

[166] Schlegel, M., Brosig, G., Eckert, A., Engelke, K., Jung, M., Polt, A., Sonnenschein, M., and Vogt, C., "Integration of discrete-event simulation and optimization for the design of value networks," in *16th European Symposium on Computer Aided Process Engineering* (Marquardt, W. and Pantelides, C., eds.), pp. 1955–1960, 2006.

[167] Schoemaker, P. J. H., "Multiple scenario development: Its conceptual and behavioral foundation," *Strategic Management Journal*, vol. 14, no. 3, pp. pp. 193–213, 1993.

[168] Secomandi, N., "A rollout policy for the vehicle routing problem with stochastic demands," *Operations Research*, vol. 49, pp. 796–802, September/October 2001.

[169] Sierksma, G., *Linear and Integer Programming: Theory and Practice*. Marcel Dekker, Inc, 1996.

[170] Stadtler, H., *Supply Chain Management - An Overview*, ch. 1, pp. 7–28. Springer, 2000.

[171] Stolyar, A. L., "Optimal routing in output-queued flexible server systems," *Probability in the Engineering and Informational Sciences*, vol. 19, no. 2, pp. 141–189, 2005.

[172] Strock, J., "Seabasing: A joint force enabler in area-denial and anti-access environments." Presentation.

[173] Sullivan, S. M., "Forward presence: Meu(soc)s in action today and tomorrow," *Marine Corps Gazette*, vol. 79, no. 8, pp. 38–38–39, 1995.

[174] Swisher, J. R., Hyden, P. D., Jacobson, S. H., and Schruben, L. W., "A survey of recent advanced in discrete input parameter discrete-event simulation optimization," *IIE Transactions*, vol. 26, pp. 591–600, 2004.

[175] Tacconi, D. and Lewis, F., "A new matrix model for discrete event systems: application to simulation," *Control Systems Magazine, IEEE*, vol. 17, pp. 62 –71, Oct. 1997.

[176] Taguchi, G., Chowdhury, S., and Taguchi, S., *Robust Engineering*. McGraw-Hill, Inc., 2000.

[177] Tang, L. and Wang, X., "An iterated local search heuristic for the capacitated prize-collecting travelling salesman problem," *The Journal of the Operational Research Society*, vol. 59, no. 5, pp. 590–599, 2008.

[178] Teypaz, N., Schrenk, S., and Cung, V.-D., "A decomposition scheme for large-scale service network design with asset management," *Transportation Research Part E*, vol. 46, pp. 156–170, 2010.

[179] Thal, A. E. and Heuck, W. D., "Military technology development: a future-based approach using scenarios," *Foresight : the Journal of Futures Studies, Strategic Thinking and Policy*, vol. 12, no. 2, pp. 49–49–65, 2010.

[180] Till, G., "Naval transformation, ground forces, and the expeditionary impulse: The sea-basing debate." Strategic Studies Institute, U.S. Army War College, December 2006.

[181] Tiron, R., "Striking a balance.," *Sea Power*, vol. 51, no. 9, p. 22, 2008.

[182] Tripp, R. S., Amouzegar, M. A., McGarvey, R. G., Bereit, R., George, D., and Cornuet, J., "Sense and respond logistics integrating prediction, responsiveness, and control capabilities." RAND, 2006.

[183] Truver, S., "'sea power 21' ... for the common good," *NATO's Nations and Partners for Peace*, no. 1, pp. 118–118–124, 2004.

[184] Tsiakis, P., Shah, N., and Pantelides, C. C., "Design of multi-echelon supply chain networks under demand uncertainty," *Indistrial and Engineering Chemisty Research*, vol. 40, pp. 3585–3604, 2001.

[185] Turk, T., "System dynamics simulation of computer networks: Price-controlled qos framework," *Mathematics and Computers in Simulation*, vol. 78, no. 1, pp. 27 – 39, 2008.

[186] Tyler, J. T., "Reality check: The trouble with scenario-based military planning," *The Brookings review*, vol. 12, no. 4, pp. 30–30, 1994.

[187] Udeanu, G., "Tactical scenarios – means of objectifying the battle field reality.," *Buletin Stiintific*, vol. 15, no. 1, pp. 64 – 82, 2011.

[188] United States Marine Corps, Washington, DC, *Amphibious Ships and Landing Craft Data Book*, August 2001. MCRP 3-31B.

[189] Urwin, E. N., Gunton, D. J., Atkinson, S. R., Daw, A. J., and deC Henshaw, M. J., "Through-life nec scenario development," *Systems Journal, IEEE*, vol. 5, pp. 342 –351, sept. 2011.

[190] Valinataj, M., Mohammadi, S., Plosila, J., Liljeberg, P., and Tenhunen, H., "A reconfigurable and adaptive routing method for fault-tolerant mesh-based networks-on-chip," *AEU - International Journal of Electronics and Communications*, vol. 65, no. 7, pp. 630 – 640, 2011.

[191] Varum, C. A. and Melo, C., "Directions in scenario planning literature a review of the past decades," *Futures*, vol. 42, no. 4, pp. 355 – 369, 2010. ¡ce:title¿Learning the Future Faster¡/ce:title¿.

[192] Vommi, V., Murty, and Seetala, S., "A simple approach for robust economic design of control charts," *Computers and Operations Research*, vol. 34, no. 7, pp. 2001 – 2009, 2007.

[193] Vuchkov, I. N. and Boyadjieva, L. N., *Quality Improvement with Design of Experiments*. Kluwer, 2001.

[194] Wallace, T. R., "Marine expeditionary brigade: Centerpiece of the future." School of Advanced Military Studies United States Army Command and General Staff College, May 2005.

[195] Wang, Y. M., Yin, H. L., and Wang, J., "Genetic algorithm with new encoding scheme for job shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 44, pp. 977–984, 2009.

[196] Warrington, L. and Jones, J., "Representing complex systems within discrete event simulation for reliability assessment," in *Reliability and Maintainability Symposium, 2003. Annual*, pp. 487 – 492, 2003.

[197] Westerlund, A., Gthe-Lundgren, M., and Larsson, T., "A stabilized column generation scheme for the traveling salesman subtour problem," *Discrete Applied Mathematics*, vol. 154, no. 15, pp. 2212 – 2238, 2006.

[198] Williams, T. W., "Meu(soc): The jewel in the crown of our corps," *Marine Corps Gazette*, vol. 78, no. 3, pp. 30–30–32, 1994.

[199] Zegordi, S. H. and Nia, M. A. B., "Integrating production and transportation scheduling in a two-stage supply chain considering order assignment," *International Journal of Advanced Manufacturing Technology*, vol. 44, pp. 928–939, 2009.

[200] Zhang, L. and Zhang, Y., "Approximation for knapsack problems with multiple constraints," *Journal of Computer Science and Technology*, vol. 14, pp. 289–297, 1999.

[201] Zhuang, Y., Shi, D.-M., Du, W.-B., Zhang, H.-F., and Wang, B.-H., "Integrating local dynamic and global static information for routing traffic on networks," *International Journal of Modern Physics C*, vol. 22, no. 6, pp. 649–659, 2011.

[202] Zimmermann, A., *Stochastic Discrete Event Systems: Modeling, Evaluation, Applications*. Springer, 2008.