A MODELING TRADE-OFF FORECASTING ENVIRONMENT FOR MILITARY AIRCRAFT SUSTAINMENT

A Thesis
Presented to
The Academic Faculty

by

Elizabeth Saltmarsh

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

Georgia Institute of Technology
May 2015

A MODELING TRADE-OFF FORECASTING ENVIRONMENT FOR MILITARY AIRCRAFT SUSTAINMENT

Approved by:

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

Professor Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Assistant Professor Graeme Kennedy
School of Aerospace Engineering
Georgia Institute of Technology

Assistant Professor John Salmon
Department of Mechanical
Engineering
Brigham Young University

Dr. Kelly Griendling
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: April 3, 2015

ACKNOWLEDGEMENTS

I would first like to thank Dr. Mavris for his guidance throughout the thesis process, especially for his help in scoping and framing the work. Dr. Schrage provided helpful information about the military sustainment context into which my specific problem fit. Dr. Kennedy's insight into the optimization studies was key in fully exploring this aspect of the problem. Dr. Salmon's advice about the model, the visualization of results, and how this related back to the overall thesis goals was much appreciated, as were Dr. Griendling's unique contributions to these areas. Dr. Griendling also contributed the idea for a use case methodology to help explain how SustainME works, which helped to ground the results chapter. Burak Bagdatli's modeling insights were helpful for finding alternative solutions to problems I initially encountered in this area, and his advice on how to visualize some of the results was helpful. Nick Molino contributed ideas for how to normalize the objective function in the optimization, and Scott Wilson was instrumental in helping to work out some of the practical aspects of the optimization problem as well as contributing visualization advice. My work with Phil Fahringer, Anne Flannigan and Heather Miller at Lockheed Martin over the years helped prepare me to understand and model military aircraft sustainment, and much of this experience was helpful in both developing and verifying the model. Finally, many members of the Aerospace Systems Design Lab and Aerospace Department support staff have contributed their assistance throughout this process by providing information and support; but Loretta Carroll deserves special recognition for her help in facilitating each step, both major and minor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

3LM Three Level Maintenance.

ABM Agent Based Model.

ALIS Autonomic Logistics Information System.

A_M Materiel Availability.

A_O Operational Availability.

CBM Condition Based Maintenance.

CBM-MiMOSA Condition Based Maintenance Mission and Maintenance Optimization of Scheduling Alternatives.

DoE Design of Experiments.

GA Genetic Algorithm.

JDIS Joint Distributed Information System.

JSF Joint Strike Fighter.

LP Linear Program.

LRU Line Replaceable Unit.

MAS Multi Agent System.

MC Mission Capable.

MFHBF Mean Flight Hours Between Failures.

MFOP Maintenance Free Operating Period.

MILP Mixed-Integer Linear Program.

MIQCP Mixed-Integer, Quadratically Constrained Program.

MTBF Mean Time Between Failures.

NMC Non Mission Capable.

OEC Overall Evaluation Criterion.

OEM Original Equipment Manufacturer.

O&M Operations and Maintenance.

PBL Performance Based Logistics.

PDF Probability Density Function.

PHM Prognostic Health Management.

R_M Materiel Reliability.

R_O Operational Reliability.

R&R Remove and Replace.

SPN Stochastic Petri Net.

Sustain-ME Sustainment Modeling Environment.

VMI Vendor Managed Inventory.

SUMMARY

Military aircraft sustainment is a traditionally difficult problem that has significant consequences if managed incorrectly. First and foremost it is an expensive problem even at the best of times, and frequently costs more money that was anticipated. Because of the uncertainty associated with so many aspects of sustainment, supplying operational military aircraft with spare parts in a timely fashion is both difficult and costly. History shows this to have been true throughout the history of military aviation.

Because sustainment has been a difficult and insufficiently solved problem in the past, current Air Force doctrine has focused on ways of improving both the performance and affordability of sustainment. In particular, a new paradigm shift has been proposed that combines several promising operational methods. The first of these are vendor managed inventory and performance based logistics, which act together to shift the responsibility for supplying spare parts for systems to outside parties, often the manufacturer that developed the systems. The goal of these two paradigms is to allow the vendor to determine the most efficient way to supply the aircraft, which should lead to savings that can benefit both the manufacturers and the military. However, this new method of operating is being introduced at the same time as another paradigm shift is taking place which makes the aforementioned goals much more difficult.

These additional paradigms are condition based maintenance and affordability based operations. Condition based maintenance is a relatively new maintenance paradigm that seeks to improve maintenance by predicting, rather than reacting to or

preventing failures. This is done through inspection or monitoring of aircraft components for signs of failure, which are then used to more accurately predict when failures may occur. Condition based maintenance generally assumes that this information will be used to delay maintenance for as long as possible without actually allowing the component to fail. This has an advantage in terms of inventory use, because parts are used to their fullest possible extent, as compared to a preventive maintenance paradigm where they are removed after a certain number of flight hours regardless of whether these components have worn out. However, it introduces a disadvantage in that part failures occur at stochastic times, whereas under preventive maintenance the times could be predicted well in advance and were planned at regular intervals.

The final paradigm's focus on affordability only complicates the issue by removing spare inventory margins. However, the stochasticity of maintenance times that is inherent to condition based maintenance makes it difficult to reduce inventory levels because some margin is needed to react to the stochastically high failure periods which will occur. This only further complicates the role of the vendor, who has been contracted to provide these spare parts based on abstracted goals. Performance based contracts require that a certain level of availability or mission requirements be met at all times, whereas the traditional supply based contracts required the military to determine up front how many spares were required. Under this paradigm shift, the vendor must determine how many spares are needed based on uncertain information, and at low cost.

If the above set of paradigms are to be at all successful together, some form of planning tool is absolutely essential. This thesis describes the development of such a tool, a sustainment trade-off modeling environment that allows decisions such as those involved in the Air Force paradigm shift to be simulated and their behavior forecasted. The modeling environment, called Sustain-ME for Sustainment Modeling Environment, was developed based on information that is available in the literature,

and the assumptions made in creating it are directly addressed within this document. Additionally, the activities performed for verifying the behavior of Sustain-ME are extensively listed to build confidence in the model and the results that are obtained from it.

Once Sustain-ME was created, it was used to answer several of the questions associated with the current Air Force paradigm shift. The fact that the paradigm shift represents a highly stochastic way of operating was hypothesized, tested, and found to be true. The behavior of condition based maintenance was particularly found to be nonstationary at the conditions associated with the paradigm shift, particularly the reduced inventory levels desired. Additionally, the question of which metrics are appropriate for measuring the performance of sustainment was addressed through additional testing. This testing showed that two of the primary metrics used, operational availability and operational reliability, do not have a simple relationship that correlates to one another at all times. Therefore both metrics are deemed necessary to explain the true behavior of a set of operational conditions.

Sustain-ME was also demonstrated for a sample use case, which compared three different maintenance paradigms to one another under similar conditions. The three maintenance paradigms are reactive maintenance, condition based maintenance, and a novel strategy for using condition based maintenance information to schedule maintenance that uses optimization to regain some of the benefits of a preventive maintenance strategy. These paradigms were compared in several categories, including the inventory required to achieve a 70% operational availability, the variability of each over time and between different repetitions of the sustainment process, and the ability of each to fly a required set of missions. The condition based maintenance strategy was found to be the most effective generally speaking, though the novel paradigm was found to perform better when the detection time for failures is low.

CHAPTER I

INTRODUCTION

Military aircraft represent the forefront of aerospace technology and are frequently tasked with operating to their limits. For any complex system operating in this manner, component failures are common and repairs must be completed in an effective and timely manner. This procedure of operating a fleet of aircraft to achieve military objectives and carrying out the necessary support activities is commonly referred to as "sustainment". Explicitly, the Joint Chiefs of Staff define sustainment as "the provision of logistics and personnel services necessary to maintain and prolong operations..."[86]. Though this definition implies a binary nature of sustainment (i.e. the aircraft is either sustained or not sustained), in reality the level of investment in sustainment leads to different outcomes for how well operations may be maintained and prolonged. The availability and effectiveness of aircraft, the affordability of sustainment, and the risk associated with operating at a level of sustainment are all priorities which must be balanced against one another to provide adequate performance with certainty at an acceptable cost.

1.1 Sustainment

From a financial standpoint a significant component of any weapon system's total life cycle cost is spent on sustainment, an estimated 60 to 75 percent, the majority of which covers supply chain costs[112]. This translates to billions of dollars every year; in 2011, the United States Air Force spent over fifty billion dollars on Operations and Maintenance (O&M) activities, with eight billion alone going to O&M for its primary combat forces [91]. Furthermore, the number is growing, not shrinking. O&M costs consistently increase by 2 to 3 percent per year after inflation[112].

From a performance perspective, sustainment is also critical, and is often not adequate to meet the evolving requirements of the military. According to the Air Force's Scientific Advisory Board, "...the sustainment enterprise consistently falls short of the targets whenever funding falls short of the requirements, which is frequent.[111]" With unnecessary maintenance activities accounting for as much as 33% of the total maintenance costs[84], this is hardly surprising. However, these issues are already well known within the military and not trivial to resolve. Past efforts at increasing affordability have had mixed success[6], and as a result it is important to gather as much information as possible before making decisions to balance cost with performance. If such efforts are undertaken without full understanding of the consequences, it is fully possible to sacrifice performance without gaining any significant affordability benefits in return.

According to the Air Force's 2013 Contract Sustainment Support Guide, two of the most broadly applicable ways for gauging the performance of sustainment are the operational availability (A_O) and operational reliability(R_O) of the fleet[117, 30]. These are, respectively, the percent of time the fleet's aircraft spent flying or available to fly missions[62, 27] and the percent of mission objectives achieved by the fleet[117]. A_O and R_O are listed as key performance metrics for several of the current "best practices" identified in the document, including condition based maintenance which will play a key role in this thesis. It should be noted that the recent DODI5000.02, also from 2013, defines material availability and materiel reliability as Key Performance Parameters (materiel availability) and Key System Attributes (materiel reliability) respectively[1]. Though this might appear at first glance as if these metrics supercede those drafted in the Contract Sustainment Support Guide from three months earlier, a bit of reading reveals two key facts. First, materiel availability and operational availability are, in some cases, defined by the same equation. The version that will be used in this thesis computes A_O and A_M as the percent of time that a fleet of aircraft

as a whole spends in an available state, or total fleet uptime divided by the total fleet downtime. Computing availability in this way seems to satisfy several of the formulative documents for Sustainment that currently dictate Air Force policy. The second key fact distilled from the literature is that materiel reliability is controlled by design rather than operational policy[27, 1]. This means that, depending on the point in the lifecycle that sustainment is being planned, R_M may be either a fixed value set by a preexisiting design, or a value that can still be changed with future design decisions. After a certain point in the design process, however, R_M will represent a fixed parameter for the aircraft rather than a changing metric that reflects the effectiveness of sustainment policy. For this reason A_O and R_O should be retained as primary metrics for evaluating sustainment.

A_O and R_O are dependent on several factors. First, the **aircraft design** plays a role: aircraft materiel reliability[27, 1] (distinct from the operational reliability of the fleet) determines the rate at which parts will break as a function of the aircraft's utilization[70], which is determined by fleet level operational requirements. **Operational requirements** are driven by whether it is peacetime or wartime and by the specific strategic objectives associated with either type of operation, such as a list of targets to be destroyed or payload to be delivered. **Maintenance paradigms** impact fleet level performance by controlling when and under which conditions aircraft are maintained; this influences the number of times each aircraft visits maintenance over a specific period of time, the likelihood of other aircraft being present when maintenance is performed, and the likelihood that an aircraft will experience part failure during a flight. Finally, **sustainment resources** in the form of personnel, equipment, facilities, fuel, weapons and spare inventory influence the rate at which aircraft can be maintained and returned to an operational status. The need to replenish these resources necessitates a **supply chain**, the effectiveness of which also impacts fleet level performance. Aside from the design of the aircraft, these influencing factors are

the major components of military aircraft sustainment, represented in Figure 1.

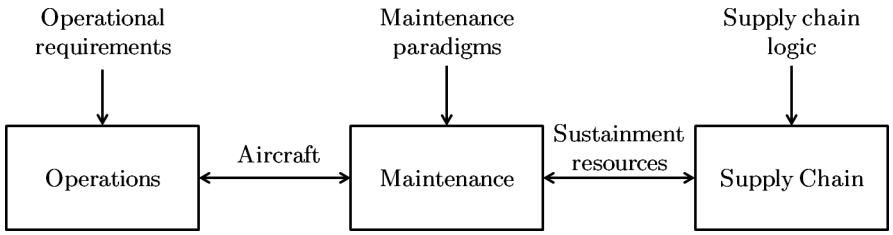


Figure 1: Sustainment components

1.2 Paradigm Shift

Given the high degree of complexity associated with sustainment combined with its importance in seeing operational benefit from a system, different paradigms have been proposed for how to operate, maintain and supply military aircraft over the years. These paradigms gain favor with different groups and individuals within the military, and as time goes on concrete evidence may be gathered to determine their effectiveness. However, this evidence gathering process takes years and is subject to the effects of political and personal motivations on reporting and interpreting data. While politics will to some degree always influence decision-making, gaining information with greater transparency and speed can help to offset biases simply by providing everyone with the same information while there is still time to affect change. One way of doing this is to use computer models to emulate the effects of different operational paradigms on the sustainment process. While such models will require some degree of abstraction to provide performance and cost estimates, they allow many of the effects and interactions within sustainment to be captured so that high level trends can be found. Useful models will provide information of the form "if A then B" which may then be applied to the real world sustainment process to make improvements or avoid pitfalls. This thesis presents a modeling environment of this type, and uses it to compare the estimated performance, cost, and risk of different sustainment paradigms.

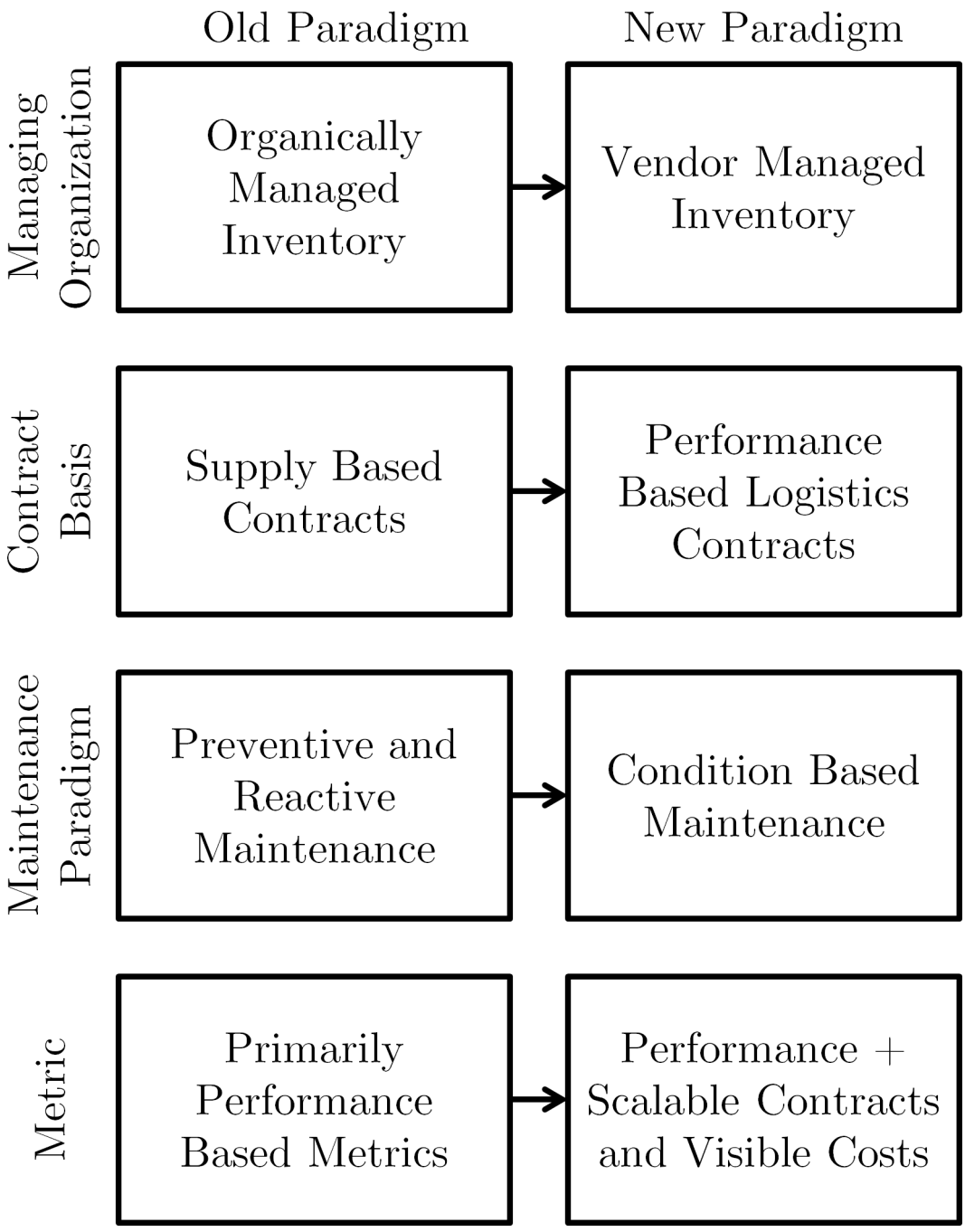


Figure 2: Air Force operational paradigm shift

In particular, this thesis is motivated by a paradigm shift that has occurred over the past few years within the United States Air Force. Figure 2 shows the traditional paradigms alongside those that replace them. The first is a shift from organic (i.e. government-owned and managed) inventory to **Vendor Managed Inventory (VMI)**, where an outside entity is contracted to refurbish spare parts when they break and manage spare inventory levels across the Air Force[30]. Another shift from supply based contracts to **Performance Based Logistics (PBL)** contracts requires the vendor to supply inventory to meet a level of performance rather than a specific number of spare parts, increasing the level of abstraction and therefore risk for the vendor. The two in combination are well-established for aircraft engines [89, 80] and helicopters[42], and have recently begun to propagate to the airframe level [100, 14, 93]. In particular Lockheed Martin is interested in adopting this business model in support of the F-35 Joint Strike Fighter[100, 79], though formal agreements currently only cover initial sustainment activities[113]. VMI and PBL together are intended to serve the purposes of both vendor and Air Force by allowing the vendor more freedom to operate efficiently, with profit emerging as the number of spares required to provide a level of performance is optimized[30], and by giving the military a guaranteed level of service at a reasonable price.

However, two additional paradigm shifts complicate the task. The military has traditionally used a combination of preventive (scheduled) maintenance to replace parts when statistics suggest they need to be replaced, and reactive (unscheduled) maintenance when these predictions do not catch failures in time to prevent them. The advantages of primarily preventive maintenance are that parts are usually replaced regularly and before they break, reducing in-flight failures and therefore risk; however, parts are also replaced before they have been fully utilized. As a result, the Air Force has recently begun to focus on using **Condition Based Maintenance (CBM)** which replaces parts when inspection or monitoring suggest they are close

to failure. CBM thus reduces the waste of parts which are still serviceable, but does so by allowing the uncertain timing of maintenance events to return. This uncertainty makes it difficult for vendors, who are removed by distance and organizational barriers from the maintenance process, to predict inventory needs and to supply those needs in a timely manner.

The final complication is driven by an increasing focus on **scalable contracts** and **visible costs** in the interest of affordability, which increases vendor risk by asking vendors to achieve the previous three goals at lower cost than ever before. This is done in part by aiming for a level of inventory where operations are in perfect balance with supply chain and aircraft availability is at a medium level, 70% according to the Air Force[111]. However, as the Department of Defense states in their 2009 "Weapon System Acquisition Reform Product Support Assessment", "[t]he ideal situation would be steady state, where resources are adequate to fully support requirements. The reality is this rarely occurs; requirements consistently exceed available resources.[112]" The reason steady state performance is usually not achieved is answered in another quote, this time from the RAND Corporation in a report prepared for the Air Force in 1993: "Parts demand processes are frequently **nonstationary** [emphasis added], i.e., the expected number of demands for a given stock number in a time period of specified length varies over time, and the magnitude and direction of that variation are almost never predictable. The problem of minimizing the number of items in long supply given a system performance goal is made extremely difficult by this uncertainty.[2]" And if demand uncertainty makes it difficult to operate with minimized inventory, this is expected to be especially true under a CBM approach.

Taken together, these paradigms suggest a nontrivial challenge for vendors, who will be responsible for achieving prespecified performance levels under highly uncertain circumstances in order to meet requirements. Failure to do so would mean

sacrificing bonuses or even paying penalties[26], so the risk associated with this combination of paradigms is high. On the Air Force's side, risks may be even higher depending on how performance shortfalls translate to military objectives, and what the military consequences of not meeting those objectives are. For all these reasons, it is important to correctly formulate contracts and planning up front to quantify the degree of uncertainty associated with sustainment decisions and reduce the impacts of this uncertainty where possible. High level calculation models are helpful for formulating initial predictions about the likelihood of success of these decisions, but may not capture enough detail to fully understand the implications of those decisions. Given the type of behavior described in the literature, particularly the word "nonstationary", more detailed models should be used to carefully examine the assumptions and decisions being made to ensure the performance of sustainment under those decisions and assumptions is adequate to achieve requirements. Furthermore, such models should be transparent and available to all parties involved in sustainment so that everyone has the best information possible to plan a cohesive and high performing operational structure. This is supported by a statement from the former Under Secretary of Defense for Acquisition, Technology, and Logistics and current Secretary of Defense, Ashton Carter: "In our country we buy our military equipment from private industry, so they're our partners in equipping our forces...I would like to have a relationship of candor and dialogue...we're in this together.[22]" If open dialogue is good, then open models are even better. At the moment, no open source modeling environment of this type exists. This thesis demonstrates how such a sustainment trade-off modeling environment, or Sustain-ME[1] for short, can be used to provide information to all decision makers early on in the contracting or planning process as a basis for making reasonable and consistent decisions.

[1]Named from Sustainment Modeling Environment.

1.3 Sustain-ME Use Case

The specific decisions made in developing a sustainment forecasting model are detailed in Chapter 2. However, it is first important to establish the intentions behind creating this environment beyond the desire to provide a sustainment planning ability to decision makers. To do this, a general use case for Sustain-ME is first described. Next, a specific example study is developed which will be used to demonstrate the utility of Sustain-ME throughout the thesis.

1.3.1 General Use Case

At a high level, Sustain-ME is designed to discover trade-offs within the design and operational strategy space by manipulating inputs and assumptions and forecasting trends. This allows the user to provide evidence to support or belie assumptions that have been made about the best way to design and operate an aircraft with sustainment goals in mind. Depending on what is changeable from the perspective of the user, different aspects of sustainment may be tested to quantify their impact on fleet level parameters[2]. From this high level perspective of Sustain-ME's purpose, a few questions can be posed which will directly impact how the environment is formulated.

Because Sustain-ME needs to capture high level sustainment behavior, the first question that naturally arises is which metrics should be captured. Section 1.1 discussed operational availability and operational reliability, two metrics that the Air Force uses to evaluate sustainment performance. However, the availability of aircraft and the goals achieved by those aircraft are not unrelated metrics. This prompts the following research question:

[2]Focusing on the fleet level allows the behavior of the whole sustainment process to emerge, rather than assuming that trends for a single aircraft scale to the fleet level. Also, focusing at a fleet level provides more information, as the aircraft level parameters can still be captured and output for each of the aircraft in the fleet.

Research Question 1: Are metrics A_O and R_O both required to capture the behavior of sustainment?

To answer this question, all that is needed are examples of the relationship between A_O and R_O changing under different conditions. Put simply, if A_O and R_O have a predictable relationship, only one metric needs to be captured to determine the impact on both. However, if the relationship between the two varies according to the specific test being run, then both metrics must be captured to form a complete picture of the behavior of sustainment. This leads to the first hypothesis for the development of Sustain-ME.

Hypothesis 1: The relationship between A_O and R_O is complex and cannot be represented by a simple correlation.

The means by which Hypothesis 1 will be tested will be discussed in Chapter 3.

Because Sustain-ME needs to capture sufficient behavior to discover where tradeoffs exist and what behavior may occur under different design and sustainment decisions, the second question pertaining to Sustain-ME concerns what type of effects must be captured by the environment. Specifically, Section 1.2 quoted a passage in a report from 1993 which highlights demand uncertainty as a source of difficulty in managing inventory[2]; Section 1.2 theorized that the nonstationary nature of part demand would continue to play a role under condition based maintenance. Given that both modern[112] and older[2] references predict the possibility of nonstationary behavior within sustainment, it seems reasonable that these effects would need to be captured by sustainment models to fully represent sustainment.

At this point it is helpful to discuss a few terms: stochastic process, uncertainty, and stationary. Lawler defines a stochastic process as "...a random process evolving

with time.[72]" Where a deterministic process will always end in the same result given the same initial state, a stochastic process may evolve in different ways over time given the same initial state. For stochastic processes, underlying random effects play enough of a role that the outcome cannot be determined without observing the evolution of the process over time. Uncertainty is frequently used to mean two different things: first, a lack of knowledge about a process (also known as epistemic uncertainty) and second, an inability to predict the outcome of a process or the state of a system (also known as aleatory uncertainty)[8]. It is this second form of uncertainty that is present in sustainment when demand fluctuations occur and which therefore makes sustainment a stochastic process. Epistemic uncertainty is also present in sustainment in that the benefits and consequences of different decisions are not always known; this form of uncertainty is the motivation for creating Sustain-ME in the first place. Finally, a stochastic process is stationary if its behavior reaches a point over time where the probability of being in a given state is constant[72]. This does not necessarily mean that the final state is *known*, but rather that the set of final states is known and the probability of being in any of them is also known. The reference in Section 1.2 to nonstationary part demand processes can thus be interpreted to mean that the long term part demand for sustainment under the conditions assumed in the paper does not exhibit steady-state behavior.

Finally, because Sustain-ME needs to incorporate enough aspects of sustainment to capture different design and operational decisions that might be made, the third question pertaining to Sustain-ME concerns which steps and processes must be modeled to represent sustainment. For instance, what supply chain elements must be included to capture the major real world effects? The answer will, of course, depend on the focus of the study. A study focusing on different supply chain paradigms might need a much more in-depth supply chain model than one focusing on maintenance

paradigms. But there is likely to be a baseline level of fidelity that must be captured in order to not miss basic supply chain behavior effects. This is generalized in Research Question 2.

Research Question 2: What level of fidelity is required to capture the major trends within sustainment?

Answering this question is nontrivial. As stated in Section 1.1, sustainment is generally composed of three interacting processes: operations, maintenance, and supply chain. Each of these processes needs to be present to fully capture sustainment behavior. However, even the basic level of fidelity required may depend on the process. Rather than forming a hypothesis, Research Question 2 will be explored in-depth in Chapter 2 and at that point hypotheses may be formed.

1.3.2 Example Study

The paradigm shift discussed in Section 1.2 involves a separation between the supply chain and the operations and maintenance portions of sustainment. In transferring responsibility for spare parts supply to an external entity, the Air Force is moving toward a policy known as "hole-in-the-wall maintenance[70]". The colloquial name comes from the fact that parts are taken off the aircraft and essentially forgotten about until they return as refurbished spares. This differs from the traditional Three Level Maintenance (3LM) paradigm[109, 30] which is discussed in more depth in Chapter 2. Though the problem of how to achieve rapid turnaround of spare parts is a significant one, as has already been stated, from the Air Force's perspective it will soon be someone else's problem. For this reason the example study used to illustrate the development and use of Sustain-ME will focus on the aspects of sustainment that the Air Force intends to control, namely operations and maintenance. The supply chain will be modeled using reasonable assumptions, but will be taken as a given for

the operations and maintenance side.

Of operations and maintenance, the maintenance process has been included in the paradigm shift in the decision to switch to condition based maintenance. Therefore the example study will focus on this decision and evaluate the performance of CBM under a variety of other conditions. However, it will also be compared to a more traditional maintenance paradigm and a novel maintenance paradigm to show the ability of Sustain-ME to represent a variety of decisions and compare them. The inclusion of a novel maintenance paradigm allows Sustain-ME to be demonstrated for maintenance paradigms that have not yet been conceptualized or tested in the real world, as this would be an important application of such an environment. For this thesis, the novel maintenance paradigm will be developed to test assumptions made in the traditional CBM approach. For CBM, many authors have assumed that the part condition information would best be used to maintain parts at the last minute, just before failure[119, 111, 48], known as just in time replacement[120]. However, this is not the only option for performing condition based maintenance; Jardine recognizes maintenance decision making as the crucial third step of the CBM process after data acquisition and data processing[60], and other authors also acknowledge that external factors may influence the best time to conduct maintenance[53, 68, 67, 122]. Given this possibility, the additional benefit of Sustain-ME can be demonstrated in comparing a new approach representing a different assumption about CBM to the traditional CBM logic. Therefore the example study will compare the behavior of a traditional maintenance paradigm, a traditional CBM paradigm, and a novel CBM paradigm under similar operational and design conditions. The logic behind the novel CBM paradigm will be discussed in Section 1.4.

Because the example study will be performed for a situation where nonstationary behavior can reasonably be expected to occur, one further research question arises. Despite the seemingly reasonable expectation that stochasticity will dominate the

region of interest under the paradigm shift, where inventory is minimized to match a target value of A_O and a CBM maintenance paradigm creates stochastic part failure times, operations research is a field that commonly returns results that could not be predicted using common sense. Nonlinearities within the model frequently yield surprising behavior, and the only way to determine this is to model the problem with a sufficient degree of detail. This is one of the motivations behind the modeling environment created in this thesis. As a result, this expecation is one of the most important tests that Sustain-ME will perform. It is formalized in Hypothesis 2.

Hypothesis 2: At the conditions cited in the Air Force sustainment paradigm shift, where minimal inventory is selected to meet a target value of 70% A_O, and where maintenance is performed based on a condition based maintenance policy, stochasticity will dominate the performance of sustainment.

The means by which Hypothesis 2 will be tested will be discussed in Chapters 2 and 3.

The remainder of this thesis will describe the logic behind, development of, and testing of a sustainment modeling trade-off forecasting environment that can serve as an open source basis for sustainment decision making. This environment will first be developed for a general case, and will then be used to implement three specific maintenance paradigms for the purpose of demonstrating how the environment may be used to facilitate trade-off studies among different sustainment alternatives.

1.4 Novel Maintenance Paradigm

The thinking behind the novel maintenance paradigm used to test Sustain-ME comes from the recognition that condition based maintenance has some unfortunate similarities to reactive maintenance, as was discussed in Sections 1.2 and 1.3.2. As discussed in Section 1.2, the uncertainty associated with when part failures occur will control

CBM under the assumption of just in time maintenance, and this will make it difficult for vendors to achieve required performance levels with certainty. Consequently, the novel CBM strategy modeled in Sustain-ME will seek to counteract this effect by performing maintenance at regular intervals, as was done for scheduled maintenance. Unlike scheduled maintenance, however, replacements will not occur until a failure signal is detected; therefore parts that are still indicated as functional are never replaced. Similar ideas have been proposed for scheduled maintenance [20], opportunistic maintenance (grouping maintenance activities based on soon-to-occur failures) [68], workforce planning [67], and machine failure [122]. A similar concept was implemented from the mission scheduling side by Iavkovidis, who asked individual officers to choose scheduling philosophies based on experience or intuition designed to allow maintenance to occur with minimal impact on fleet performance[56]. However, as Iakovidis discusses, these philosophies are not rigorously determined nor scientifically tested. This thesis seeks to develop a new strategy based on mathematical principles, which will shift the balance between unscheduled and scheduled maintenance benefits and drawbacks toward a region that is more stationary, but less wasteful. The means by which this will be accomplished will be discussed in Chapter 2.

1.5 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 discusses several topics from the literature that will inform the development of Sustain-ME. Chapter 3 takes this information to inform the development of experiments to test the research questions and hypotheses developed in this chapter. Chapter 4 discusses the development and testing of Sustain-ME, as well as the results of the experiments developed in Chapter 3. Chapter 5 then uses Sustain-ME to perform the example study established in Section 1.3.2. Finally Chapter 6 draws conclusions and discusses contributions and future work.

CHAPTER II

BACKGROUND

Chapter 1 introduced a paradigm shift in military aircraft sustainment and discussed how modeling and simulation could be used to evaluate the behavior of sustainment under new paradigms. This chapter looks at topics from the literature which may contribute to the development of a modeling environment intended to represent this paradigm shift, Sustain-ME. The first of these is maintenance philosophy, which encapsulates the variety of ways maintenance may be performed and the reasons these methods are chosen (Section 2.1). Next, the literature on an enabling technology for condition based maintenance, prognostic health management (PHM), is reviewed to provide additional insight into how CBM may work (Section 2.2). Next supply chain management is described since, though this field is tangent to the specific example study performed in this thesis, the supply chain informs the work being done here (Section 2.5). Next the details of sustainment that must be included in Sustain-ME will be researched in order to help answer Research Question 2. Maintenance metrics will then be defined in more detail (Section 2.4), after which optimization and decision-making methods will be reviewed since these will inform the novel approach to performing CBM used to demonstrate Sustain-ME's abilities (Section 2.6). Finally this information will be synthesized to inform the discussion on modeling type (Section 2.7).

2.1 Maintenance Philosophies

Bateman states that there are three kinds of maintenance: reactive (also known as unscheduled or corrective), preventive (scheduled), and predictive[11, 36]. Although

predictive maintenance through Condition Based Maintenance has already been introduced as an important aspect of the Air Force sustainment paradigm shift, an explanation of where it fits into the larger set of maintenance philosophies will be helpful. Next two alternate maintenance paradigms will be discussed. Though these maintenance paradigms are not the focus of the paradigm shift that motivates this thesis, and are therefore not modeled using Sustain-ME, it would be fairly straightforward to implement these alternate policies using Sustain-ME. How this might be achieved is also discussed.

It will also be helpful to point out that, though the risk levels and enabling methods associated with each may be different, they are all designed to efficiently keep systems running. As a result, the maintenance steps that are carried out will largely be the same between the different philosophies. The primary change will always be the basis for the decision to perform maintenance, even when small adjustments to the steps are necessary. As a result Condition Based Maintenance practitioners can and should learn from the experiences gained over the history of aircraft maintenance under different philosophies. Next the military strategies of Three Level Maintenance and Hole-in-the-Wall maintenance are discussed to augment what is known about the paradigm shift toward vendor managed inventory. Finally some reliability theory must be discussed, specifically the bathtub curve trend and what it means for Condition Based Maintenance.

2.1.1 Unscheduled Maintenance Alone

Unscheduled maintenance, also referred to as corrective maintenance, is the most basic maintenance philosophy when applied alone. This is because it requires the least amount of effort to implement: the system is used until it breaks, and then fixed so the process can start over[106]. This philosophy does not require awareness of the system's inevitable future breakages, but if awareness exists, budgeting for

maintenance or setting aside maintenance facilities and supplies can mitigate the risks. Any person operating a vehicle without paying attention to standard upkeep uses this maintenance philosophy. This policy is also common in factories; according to Bateman more than 50% of factories use pure unscheduled maintenance[11]. Though it may be the simplest strategy, unscheduled maintenance is also the least responsible since there is a chance of ruining the system. This applies even more so when the risks are compounded, either through more aggressive use (such as the stresses on an individual racing vehicle) or when an individual or organization is responsible for multiple systems (such as a rental car company or airline).

Aside from simplicity, the major benefit of unscheduled maintenance as a strategy is that lower maintenance cost may theoretically be achieved[106]. This reasoning is based on the notion that maintaining a system before failure, as with scheduled maintenance, wastes the usable life remaining to the part or fluid. Such reasoning is most valid when the consequence of failure is low, such as when a car's battery dies or its headlights burn out. These problems may be nuisances for the user, but do not cause any harm to the car under normal circumstances. The reasoning becomes problematic when allowing the system to operate to the breaking point causes damage to expensive components (or the system as a whole)[11]. If a car's tires are used until bald, this risks an accident that could total the car. When air filters or oil are not changed regularly, this can damage the engine and lead to more expensive repairs down the line[84]. It is because of this second situation that scheduled maintenance combined with unscheduled maintenance is the standard maintenance philosophy for many systems.

2.1.2 Scheduled Maintenance With Unscheduled Maintenance

Scheduled maintenance, or preventive maintenance, requires that the system be maintained at fixed intervals based on usage hours or real-time hours[106]. It can be a much

more effective maintenance philosophy when intelligently combined with unscheduled maintenance. This means that, when appropriate, a scheduled maintenance policy is implemented to proactively handle system degradation through use. The default, unscheduled maintenance, is used whenever unexpected failures occur or when scheduled maintenance is not deemed appropriate. Determination of when scheduled maintenance is appropriate may depend on the risks of component failure, as with the case mentioned in Section 2.1.1, or on the cost of components. When components are inexpensive it quickly becomes worthwhile to replace them if doing so can potentially avoid more costly repairs later on[23]. Cheung also examines how to mitigate the disadvantages of scheduled maintenance in production lines by manufacturing safety stores to sell while systems are offline for maintenance[21]. Unfortunately this is not an option for systems which must be constantly operational, such as military aircraft.

An additional advantage of increasing the amount of scheduled maintenance is the increased predictability of maintenance timing, which may be more convenient than repairing the system whenever a failure occurs. This is especially true when including opportunistic maintenance[87], which carries out multiple repairs at once wherever possible, reducing the frequency with which maintenance must occur. For individual systems this advantage may not be significant, but for groups of systems this advantage is critical. Airlines are a prime example of this fact, since they schedule operations to get the most use possible out of aircraft[96]. Unexpected events tend to have far reaching effects due to the high utilization rates and tight schedules of passengers, air traffic resources, and airport resources. As a result, predictability through a carefully executed maintenance policy is necessary to keep this system functioning properly. However, airlines are able to pass on the cost of such predictability to customers; the military must work within its budget and satisfy government oversight bodies that such expenditures are necessary. This makes it difficult to take full advantage of opportunistic maintenance policies.

As Section 2.1 mentioned, maintenance scheduling policy is separate from the steps that are taken to carry out maintenance. Both scheduled and unscheduled maintenance remove parts from the system, dispose of broken parts or send them to be fixed, and install a new part to replace the broken one[106]. In addition to these basic maintenance elements, repairs may be documented, the system may undergo additional servicing through cleaning or fluid replacement, and the system may be checked for correct operation after repairs have been completed. Unscheduled maintenance must further undergo a troubleshooting process before the system can be repaired since the exact cause of the failure is unlikely to be known[39]. This introduces additional risk to maintenance due to the possibility of misdiagnosing errors and continuing to operate the system with a latent failure in place. This possibility is another reason why unscheduled maintenance should be reserved for parts with a low chance of causing total system failure.

Though general guidelines for the balance point between scheduled and unscheduled maintenance have been discussed, in reality this balance will be found through a combination of computation and heuristics. Computation can help to determine the appropriate times to carry out scheduled maintenance to achieve a certain rate of failure prevention[106], then heuristics might be used to determine which parts are worth including in scheduled maintenance and which are not. The main point, however, is that this balance represents a trade between risk, cost, and convenience which is fully in the hands of the individual or organization responsible for decision making. However, there is another option for dealing with the uncertainty inherent in maintenance: condition based maintenance.

2.1.3 Condition Based Maintenance

Condition Based Maintenance (CBM) is a method for determining maintenance times based on the condition of components. This method is predicated on the assumption

that the condition can be measured[49]. The simplest version would check a system's components periodically and determine whether they are operating correctly or have failed, replacing only failed components[57]. More complex versions include the ability to determine whether a part has degraded partially, indicating a spectrum of operation between working and failing. Targeting the correct point on this spectrum allows part replacement to be carried out just before component failure. Still more complex versions of CBM give the ability to perform these checks in real-time from sensors installed throughout the system[49], though installed sensors are often paired with regular checks for other signs of degradation[36]. These increasingly complex abilities naturally come with added cost of system development and, in the case of sensor-aided CBM, the potential for new failure modes as a result of additional parts in the system. Because more sophisticated maintenance cannot be implemented for free, such strategies should be tailored to the needs of the system. Just as increasing system complexity or consequence of failure justified adding scheduled maintenance, the system must be more complex or failure must have higher consequences still to make CBM worthwhile.

While condition based maintenance will ideally derive advantages from both scheduled and unscheduled maintenance, gaining the proactive aspects of scheduled maintenance and the full component utilization of unscheduled maintenance, it does so by accepting unscheduled maintenance's fully stochastic timing on when repairs occur[106]. As stated previously, this can be problematic at a system-of-systems level when many systems are failing stochastically yet the need for these systems to be operational is constant[11]. The instability in failure times can be mitigated to some extent by holding regularly scheduled condition checks and maintaining only during these intervals when it is clear a part will soon fail. This requires part condition to be detectable along a spectrum from operating to failed, with the condition evolving slowly enough

to provide sufficient time to detect failures in advance. It should also be noted that repairing only at regular intervals retains scheduling stability but sacrifices some usable part life remaining.

Consequently, even condition based maintenance represents a balancing act between two naturally opposed goals: predictability in maintenance scheduling and maximum use of resources. Jardine discusses this directly, saying "When is the best time? That depends on your overall objective. Do you most want to minimize costs or maximize availability? Sometimes the best preventive replacement time accomplishes both objectives, but not necessarily."[61]. Where it makes up for the additional cost of development, however, is in the additional information it provides to maintainers: scheduled maintenance had no way of knowing whether maintenance was necessary at the time it was scheduled, but CBM's additional knowledge allows maintenance to be intelligently delayed[106]. In fact, CBM could allow a company to implement something that resembles traditional scheduled maintenance but with fewer unnecessary part replacements. The freedom provided by this information is helpful in allowing maintenance to flexibly react to changing needs, but must be leveraged correctly so the the best decisions possible are made.

2.1.4 Alternate Maintenance Paradigms

Two alternate maintenance paradigms are discussed here. They represent different options for increasing sustainment performance in fairly different ways. The first paradigm is the Maintenance Free Operating Period (MFOP), which replaces the traditional Mean Time Between Failures (MTBF) as a reliability metric for the aircraft[71]. The goal of choosing this as the metric is to increase the period of time that a system can operate without requiring maintenance activities to be performed, or realistically the period of time where such a thing occurs with a high degree of probability. Technically both MTBF and MFOP are functions of the design plus

the sustainment process as a whole[74], since aircraft level reliability metrics tend to be computed in flight hours, while MTBF and MFOP are computed in operational (real world) hours. However, both metrics are more closely tied to design than A_O and R_O, which are based on a higher level of abstraction. Furthermore, MFOP is much more closely tied to design than CBM, which is tangentially related but not directly controlled by the inherent parameters of the system. To achieve the goal of increasing the time during which maintenance is not required, both design and architecting improvements as well as maintenance strategy through a lifing policy are required[94]. Design is required to increase the reliability of the system sufficiently to ensure that the desired MFOP is possible, and lifing policy is required to replace aircraft that are predicted to fail within the next MFOP. The MFOP policy requires prognostics[94], making it a subset of condition based maintenance without a just in time replacement policy. To model a MFOP strategy in Sustain-ME, there are two options. First, if the assumed design reliability were good enough, the lifing policy could be programmed into Sustain-ME to attempt to achieve a desired level of MFOP. However, if the reliability is not sufficient on its own to achieve the desired MFOP, some form of feedback would have to be used to adjust the design variables by the required amount.

The second paradigm is phase maintenance. Phase maintenance is similar to scheduled maintenance, but requires a more involved set of repairs[76]. In essence, the aircraft is being thoroughly checked, refurbished, and rebuilt where necessary. The decision point for performing phase maintenance is based on the number of hours the system has spent in use[31]. Gaguzis found it to be inferior to condition based maintenance in his master's thesis, although this observation is limited to the particular scenarios he studied[43]. If the maintenance strategy were of interest, however, it could be implemented within Sustain-ME. First, the phase maintenance period would have to be specified. Each aircraft in Sustain-ME would then track its

flight time since the last phase maintenance event and submit itself for a thorough set of repairs once that period has elapsed. To simulate the time required to perform phase maintenance, information would have to be provided to Sustain-ME about how long phase maintenance takes, and what resources are required to carry it out. With this specified, phase maintenance could then be enacted by Sustain-ME either by itself, or in concert with other maintenance strategies.

2.1.5 Three Level and Hole-in-the-Wall Maintenance

Three Level Maintenance (3LM) is the current Air Force Standard [109, 30], divided between local and off-site locations. Local maintenance is divided between the first two of the three eponymous levels: organizational level and intermediate level maintenance. Organizational level maintenance includes simple actions designed to quickly turn an aircraft around to available status, such as Remove and Replace (R&R) activities which replace broken Line Replaceable Units (LRUs) with spares from inventory[70]. Intermediate level maintenance involves more complex activities but is usually restricted to standard activities which can be carried out with limited supplies available at a base[109], primarily major item replacement and refurbishment of LRUs[70]. During scheduled maintenance, which occurs at the intermediate level, other flight systems may also be overhauled and alignments or adjustments carried out if they are needed. Finally, the depot is the third level in Three Level Maintenance, and may be located at military logistics centers or contractor facilities [109]. During depot level maintenance major overhauls and rebuilding are performed, such as system level replacement.

Under 3LM, the military is still responsible for the majority of the maintenance and supply chain activities. This suggests that the new Air Force sustainment paradigm shift is inconsistent with 3LM as it has been performed in the past. What vendor managed inventory truly starts to look like is a maintenance hierarchy called

Hole-in-the-Wall maintenance[70], where the only local maintenance activities performed are R&R actions on LRUs, after which broken parts are passed to the Original Equipment Manufacturer (OEM) or separate vendor for refurbishment. Under Hole-in-the-Wall maintenance, the supply chain is moved out of military hands in accordance with the Air Force paradigm shift, meaning that the three levels of 3LM no longer apply.

2.1.6 Reliability Models

Predicting reliability is an important part of sustainment, and will be even more so under condition based maintenance since failures or near-failures are the most common source of sustainment activities when using CBM. One of the most commonly used reliability models is the bathtub curve[66], shown in Figure 3, which represents the change in the rate of failures over time. The initial portion of the bathtub curve is known as the infant mortality region. It is characterized by a decreasing failure rate, caused by the discovery and resolution of initial problems with a fielded system. This may occur through small-scale redesign, material changes or an adjustment to operational guidelines. However they are accomplished, these changes will slow and eventually stop as improvements become marginally harder. After this occurs the failure rate will be constant for a time, potentially for much of the system's life. This constant failure rate region is the one most commonly represented in elementary reliability models since it can be modeled using an exponential distribution[52], which has several mathematically useful properties. Also, since the exponential failure model represents the largest portion of a system's lifecycle any policies based on this assumption will be valuable over this period. The constant failure rate disappears near the end of the system's life once the effects of aging begin to show. Over this final region of the bathtub curve the failure rate increases as degradation leads to a growing list of system problems. At this point the system has nearly reached its usable

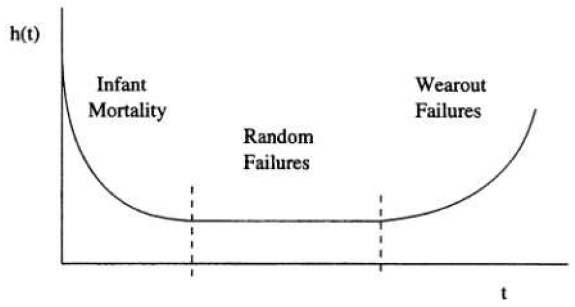


Figure 3: Hazard rate function – bathtub curve[66]

life and will soon be scrapped.

If a system is known to display a bathtub curve, this is helpful information since it reminds maintainers and operators to watch a new system carefully and to include additional repairs in early planning. However, Klutke et. al. caution that this model is somewhat in contention, and that there may not be empirical evidence supporting its validity[66]. Even if it were known to exist, there is no way of knowing the exact failure rates that will be experienced by a system ahead of time. Predictions may be made based on individual component testing and the amount of redundancy built into the system, but uncertainty is unavoidable in this process – otherwise infant mortality would not occur. Since the exact shape of the bathtub curve cannot be known until after its effects have been felt, the utility of this information is limited. Furthermore, most of the work done in sustainment will occur regardless of which portion of the lifecycle the system is currently experiencing. And so far as condition based maintenance is concerned, the bathtub curve might as well not exist since its sensor readings operate on the system's condition, not mathematical models predicting its condition. Though the technology through which CBM is implemented will most likely experience its own infant mortality and aging periods, as stated before these will not necessarily be clearly predictable in the moment and as a result they should not impact the logic behind CBM.

2.2 Prognostic Health Management

Prognostic Health Management uses a suite of sensors installed throughout the aircraft to detect off-nominal conditions that suggest failures will occur. Each sensor is tailored to a particular aircraft part based on known fault modes; these are signaled by previously identified precursors which are based on monitorable system properties [88, 100]. Internal logic then processes these sensor readings to determine whether a part failure will soon occur[119]. The ideal system would, based on a signal, be able to identify exactly when a component's performance begins to degrade and use this to predict the exact time at which it will fail. In reality, the signal being monitored will be noisy, making identification difficult and increasing the probability of the system experiencing false alarms. As Malley determines in his thesis [78], the true PHM will have to trade these two goals since the same mechanism that leads to better detection also increases the risk of experiencing a false alarm. Additionally, data for detection lead time will give only a probability measure of the remaining part life once failure has been signaled, not an exact part life remaining.

Multiple authors have modeled PHM's impacts at a fleet level[78, 39, 123], but many use a simplified approach in which the PHM is assumed to predict failure at some user input percentage of the part life, or at a distribution around a percentage of the part life [123]. One author even assumes that the PHM will be able to predict failure with sufficient time to order any spare parts that are necessary before the parts actually fail, allowing maintenance to be carried out the moment a failure occurs while only keeping on site the parts needed for detected failures[123]. This last approach can be thought of as an idealized version of CBM under PHM which is useful for providing an upper bound on the effectiveness of any method which uses PHM enabled CBM as part of a larger goal. However, the true utility of CBM at a fleet level and any methods based on it will need to take into account the eventual degree of accuracy of the system.

Those authors that do model the PHM more fully use some variation of an algorithm that analyzes a sensor signal to determine whether the signal indicates failure, and if so how soon. Elwany and Gebraeel modeled sensor data as a stochastic process with linear and exponential degradation models and brownian error terms[37, 38]; Gebraeel and Lawley also explore Bayesian and neural network approaches for updating residual-life distributions[45, 44]. Doksum and Hoyland use Wiener processes to model the degradation[32], and Kharoufeh uses Markov processes[63] and semi-Markov processes[64]. Swanson describes using Kalman filtering to process the signal[105]. Wu et al. examine two-phase degradation models, in which the presence of a signal is necessary to recognize failure, but not sufficient[121]. After failure signs are detected, inspection must confirm their validity. Akturk and Gurel incorporate usage conditions into the prediction for failure[4]. Bae et al. forumlate a method for forming new models by adding or multiplying simpler ones to create specific lifetime distributions[9]. Chinnam explores polynomial regression as a method for predicting failure[19]. Meeker et al. describe how accelerated testing can be used to estimate the degradation predictions from testing data[83]. Malley uses neural networks to analyze a signal, then creates look-up tables of distributions for use in his broader fleet model[78]. Finally, Chin describes how the specific properties of engine failure at high temperatures can be modeled to create prognostics[18]. What these approaches have in common is that many signals are obtained over time through direct observation and knowledge of the system, or these signals are notionally generated, and the signal is then used to predict at what point a part will fail. Often new signals are used to update the system and allow it to "learn" from new operational data. However, with the exception of Malley's work, these papers do not study the length of time over which the signal will evolve as failure occurs, nor predict the operational time during which detection can occur. While the detailed approach is helpful in implementing practical PHM, for the purpose of a simulation where the part failure data is notional

anyway, these methods provide additional complexity without adding value.

What is of interest for an operational model is information about when failures can be detected, and only Malley discusses this in any fashion. Since his PHM is modeled as part of a broader sustainment simulation, he assumes that a failure signal will begin to appear at 90% of part life, and that the detection will occur at 95% of part life[78]. This is a much different assumption that Yager made, where detection time was assumed to be sufficient to order parts and have them arrive before failure occured[123], an effective detection time on the order of a few months. In reality, PHM signals as a percentage of part life will most likely vary between these two extremes from part to part, as will part life itself.

One final aspect of PHM is worth mentioning. For many of the PHM systems that exist today, the prognostic information is confined to individual subsystems or systems — rarely is it integrated to the equivalent of the fleet level. However, Lockheed Martin is developing the ability to do so with its Autonomic Logistics Information System (ALIS), which integrates a PHM with their Joint Distributed Information System (JDIS)[100]. The JDIS is a network which interfaces with the entire fleet, integrating the information provided by the PHM for each aircraft[123]. This capability allows the ALIS to go beyond traditional PHM systems by automating aspects of maintenance which do not require human involvement including troubleshooting problems, ordering parts, scheduling maintenance events and completing operational checks. The goal is to free maintenance personnel for other tasks, potentially to a degree that overall fleet availability is improved. Alternately, maintenance personnel may be reduced slightly, lowering costs. Either way, the JDIS enhances the PHM significantly by providing the ability to know at any given time the fleet's operational status as well as information about degraded components which are close to failure[99]. Under a system like ALIS, new strategies for using PHM information (such as the one that will be modeled in this thesis) are especially easy to implement

because they may be programmed into the fleet manager program and can account for overall fleet level behavior patterns. Thus the system as a whole can make suggestions about the best way to carry out maintenance activities after accounting for all the failure information from the fleet.

In conclusion, the PHM literature provides a wealth of information about the different methods by which sensor signals can be analyzed to predict failure. Specifically, signals are analyzed in some fashion to provide a reading of nominal or off-nominal, and if the signal is off-nominal an additional prediction about whether and when failure may occur is generated. Depending on the sensitivity of the algorithm that analyzes the signal, the balance may swing toward missed detections or false alarms. However, the literature provides very little information about when during a part's life this signal is likely to evolve past a nominal value, which is of much greater consequence at the level of fleet behavior. The studies that have focused on fleet level behavior have made assumptions about when failure detections will occur, but do not provide insight into values for real systems. Therefore Sustain-ME will explore a range of values for PHM effectiveness, as well as a range of part reliabilities on which detections are based. Since the detection details are less relevant to the broader sustainment process, these will be simplified to a surrogate for this behavior which retains the stochasticity of the detection time but is otherwise a simple function of the part reliability. False alarms and missed detections will not be modeled, because though they impact the effectiveness of a PHM, they have little impact on decision making. Finally, the novel CBM strategy used as an example of Sustain-ME will assume that some form of JDIS is available to integrate fleet level information together and make decisions based on it.

2.3 Sustainment Details

Up to this point, the thesis has focused on the broad strategies for sustainment, and more speficially different maintenance strategies. This section explores the steps involved with each segment of the sustainment process introduced in Figure 1. Because the specific steps that are included as well as the flow between them vary for the different maintenance strategies that will be modeled in this thesis, several different versions of sustainment are presented in Section 2.3.1 through 2.3.3. Each one links back to the overall segments of sustainment presented in Figure 1, in part to show that sustainment remains the same process when viewed at a high level. The specific changes related to each maintenance strategy will be discussed, and the strategies that will be modeled using Sustain-ME will be highlighted. These maintenance event breakdowns have been created by synthesizing several sources from the literature[59, 6, 39, 123, 20, 109, 110].

2.3.1 Reactive Maintenance Steps

The first version of sustainment to be described in detail is with reactive (unscheduled) maintenance. Figure 4 shows the major steps of sustainment under this paradigm. For operations, three major steps must be completed (shown in blue in Figure 4): first, fleet personnel prepare the aircraft to fly a mission. Next, the mission is flown, and finally, mission recovery actions are performed. Both mission prep and mission recovery have several substeps, and these will be introduced in Chapter 3. Under reactive maintenance, failures will generally occur during operations. The orange boxes in Figure 4 show the steps associated with failures under a reactive maintenance paradigm. When failures occur, the aircraft will either be able to continue with the mission, or have to abort. Only aborts are pictured in Figure 4 because Sustain-ME makes the simplifying assumption that all failures during a mission cause aborts; this is due to a lack of data in the literature about the comparative rates of failures and

critical failures, critical failures being failures which require maintenance immediately. Therefore Figure 4 actually represents sustainment if all failures are critical failures.

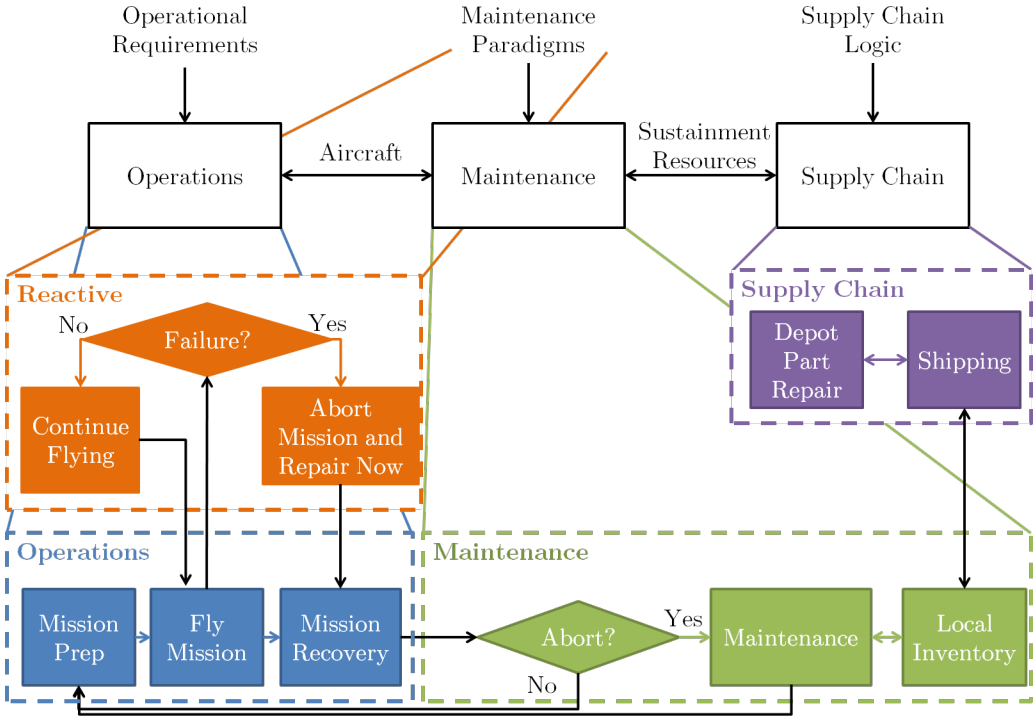


Figure 4: Sustainment steps under reactive maintenance strategy (all failures assumed critical)

Once the mission has been flown and either completed or aborted, the green steps in Figure 4 show the major maintenance steps. If a critical failure has occurred and the mission has therefore been aborted, maintenance is performed. If a critical failure has not occurred, the aircraft returns to an available status where it is once again able to fly a mission. The maintenance process also involves querying the local inventory to determine if required parts are on hand; if they are not the aircraft must wait on the supply chain, shown in purple in Figure 4. Chapter 3 will discuss the specifics of the part ordering strategy used in Sustain-ME, as well as the individual steps of maintenance.

2.3.2 Condition Based Maintenance with Inspection Steps

The second verison of sustainment described in detail is with a CBM paradigm, shown in Figure 5. Since CBM relies on diagnostic or prognostic information but does not specify how this information will be obtained, the first version of CBM presented uses inspection on the ground to determine whether maintenance must be carried out. Under this paradigm the operations steps remain the same as for reactive maintenance. Also, since in air failures can still occur under a CBM paradigm (though they occur less frequently), the steps associated with failure remain the same. Where the difference lies is in the maintenance steps, which now must include an inspection to see if forthcoming failures can be detected by signs of wear on the aircraft's parts. If the aircraft was not aborted due to failure and inspection does not find any signs that failure is needed, the aircraft can return to an available state; otherwise it must complete the steps of maintenance as before. Also, under this paradigm, the supply chain behavior remains unchanged from unscheduled maintenance.

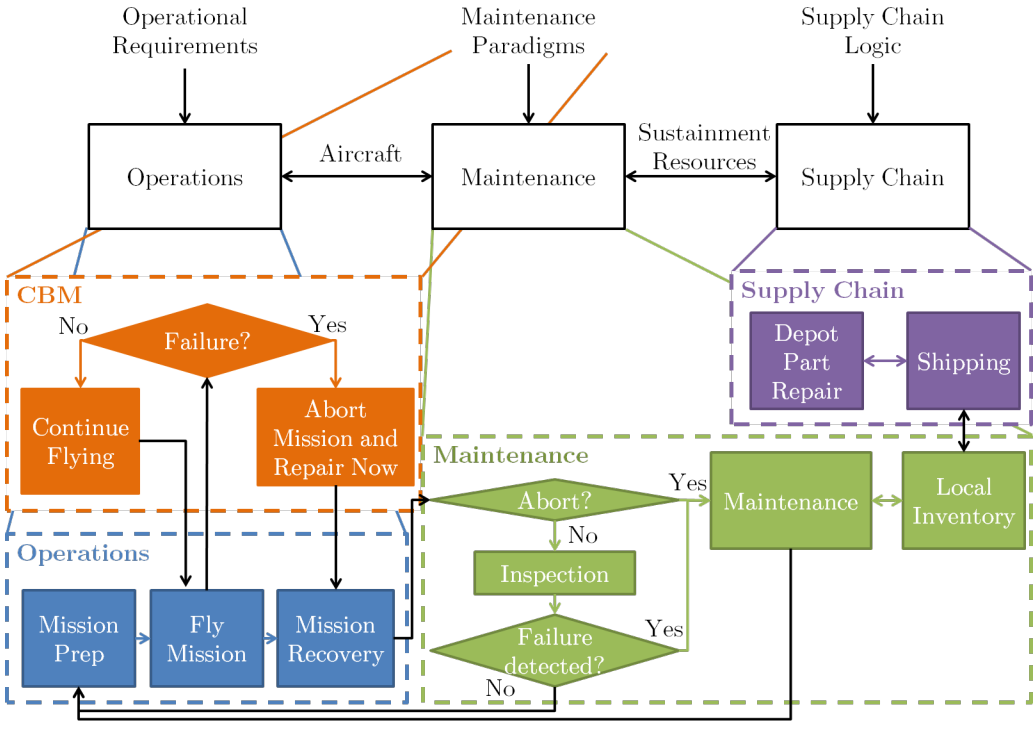


Figure 5: Sustainment steps under inspection enabled CBM strategy

2.3.3 Condition Based Maintenance with PHM Steps

The third version of sustainment described in detail is for a CBM paradigm with a prognostic health management system equipped, shown in Figure 6. Under this paradigm, operations, maintenance, and supply chain look the same as they did under reactive maintenance. In this case, the aircraft is monitored for upcoming faults automatically while in operation, meaning that inspection activities are unnecessary. (In reality, a combination of the two will most likely be used for different types of parts and part failures, but the explanation is simpler when assuming that information comes purely from the PHM system.) However, as with CBM under inspection, the possibility of unexpected failures can never be completely eliminated, and aborts will still occur. It is also possible for the PHM system to detect a failure while on a mission and predict that the failure will occur before the mission is over; though this is distinct from an unexpected failure that the PHM is unable to detect, the overal aircraft behavior is the same. Thus, under CBM with PHM, there are now more ways for aborts to occur, although depending on the properties of the PHM prediction they should be less likely.

2.3.4 Sustainment Conclusions

The sustainment approaches detailed in the literature will form the basis of the sustainment methods tested in Sustain-ME. The goal of these tests will be to establish how the different sustainment activities between the methods propagate to high-level sustainment metrics. Also, the novel maintenance paradigm developed in Chapter 3 will be based on these existing paradigms, with small changes added to shift the behavior more toward the scheduled maintenance side.

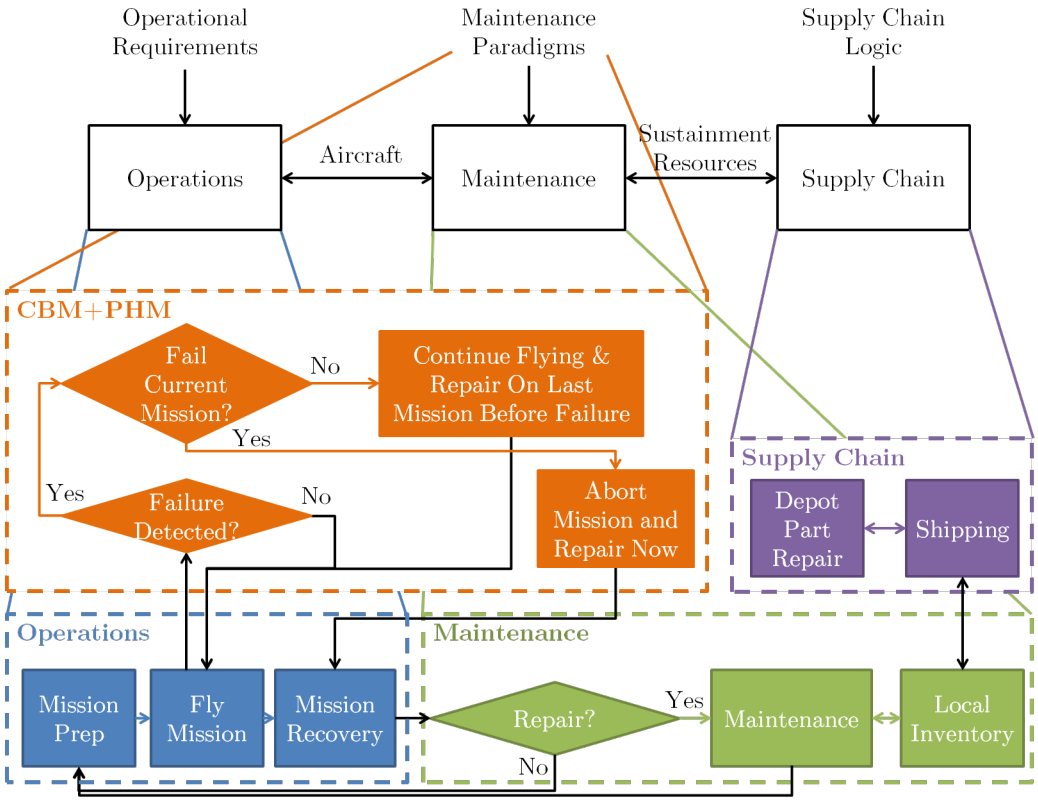


Figure 6: Sustainment steps under PHM enabled CBM strategy

2.4 Maintenance Metrics

As stated in Chapter 1, the two main maintenance metrics used in this thesis are operational availability and operational reliablity. This does not mean that additional maintenance metrics could not be captured with Sustain-ME; the model could realistically capture many maintenance metrics. However, in the interest of focus, this thesis limits itself to two. Equation 1 shows the computational basis for the version of operational availability used in this thesis.

$$A_O = \sum_{Fleet} \frac{Time_{Available} + Time_{Flying}}{Time_{Total}} \tag{1}$$

A_O indicates how often systems are Mission Capable (MC) as opposed to Non Mission Capable (NMC). However, it is sometimes a misleading metric because medium

values of operational availability may translate to 100% performance in other metrics. This is because no system can avoid maintenance entirely, and any time spent in maintenance counts against operational availability whether the aircraft was needed at the time or not. As a result this metric does not directly measure quantities which the Air Force may care the most about. However, the metric cannot be ignored either; contracts have traditionally dictated a set level of operational availability as the performance metric to meet and this may continue to occur[15, 117].

Operational reliability has also been traditionally used in contracts, and focuses more closely on the true behavior of interest for sustainment[117]. R_O is computed as a percent of mission objectives met. These may include objectives such as percent of targets killed or percent of missions flown. For the purpose of this thesis, which focuses on peacetime operations, no modeling of targets or mission success rate needs to occur. Therefore the percent of missions flown is the most appropriate version of R_O for assessing sustainment behavior. It will be computed as shown in Equation 2.

$$\%MissionsFlown = \frac{\#MissionsRequired}{\#MissionsFlown} \tag{2}$$

2.5 Supply Chain Management

Supply chain management studies the processes that move goods from one place to another, usually from a manufacturer to the end customer. True supply chains incorporate raw materials purchases, item storage at different stages of the supply chain, and transportation along the entire chain[104]. The military sustainment supply chain tends to include only a few links: the military, as the end customer, often deals directly with the manufacturer of parts or interfaces through a single contractor[30]. Supply chain management as a field seeks to answer questions about style of supply chain that best applies, how to set up the different nodes to ensure the best performance, how to most efficiently transport goods, and how to maintain adequate

inventory. For military sustainment, questions of where to place depots for foreign operations may also come into play since tariffs, export restrictions and politics will place limitations on these locations.

Due to the stochastic nature of supply chains, supply chain management has long dealt with nonstationary behavior. A central supply chain problem is handling what has been termed the bullwhip effect, where fluctuations in demand create large scale oscillations further up the supply chain. One of the biggest contributors to this effect is having long supply chains with limited communication between the organizations[73]. This should make it less of a concern for the military sustainment process which as described has only one or two links, although the Air Force states it as a problem to be addressed[30]. And in fact, Lee et. al. also identify demand distortion as an effect which operates even for supply chains with only two links[73]. In addition to these propagating effects, it should be noted that the bullwhip effect's original source is fluctuating demand, something the supply chain management field sees as an unchangeable aspect of their problem. The novel maintenance approach tested with Sustain-ME in this thesis takes a different approach by targeting demand fluctuation directly through the selection of more regular intervals for maintenance. The purpose of proposing and testing a novel approach such as this is to demonstrate how Sustain-ME may be used to evaluate new maintenance paradigms and compare them to established standards.

When it comes to the supply chain, the most important parameter in achieving the goal of minimum sustainment cost is the inventory required to match contractual levels of fleet performance. At first glance this would seem to be purely an aspect of the system's reliability. However, when viewed as a supply chain problem it becomes clear that to achieve a satisfactory level of orders filled the manufacturer must supply spare parts in excess of the demand for them. This excess supply is in part due to the lead time required to produce and ship spares [6], but is compounded by the

fact that such complex systems have many spare parts, a high portion of which fail only rarely if ever [3]. However, when components fail, spares must be available to replace them since the time required to manufacture the spare would be unacceptable. Yet producing spares for each component on the aircraft, many of which will not require replacement for years, is extremely costly. In fact, the U.S. Government Accountability Office wrote a report on the wasteful ordering of 7.1 billion dollars worth of unneeded spare parts by the U.S. Department of Defense between 2006 and 2008 [115]. As additional parts which fail more often are incorporated, the problem quickly becomes ill-defined and difficult to predict. Add in the fact that these inventory levels must sustain an entire fleet of aircraft, each with uncertain needs in terms of spares over the lifetime, and it quickly becomes clear that minimizing inventory cost is not a simple matter[62]. This is a part of the reason that this thesis seeks to model the behaviors associated with sustainment early, in order to determine which approaches work within the constraints established by the military.

2.6 Optimization and Decision Making Methods

Chapter 1 introduced the idea behind a novel maintenance paradigm. In Section 1.4, the idea of performing maintenance at more regular intervals while taking into account information from the PHM was described, but the details were left for later. Though the full formulation will be developed in Chapter 3, an initial exploration of the logic behind this approach will be briefly described here to justify the need for optimization and decision making methods. Reference has already been made to the fact that CBM's purpose is to reduce the part life wasted under a scheduled maintenance paradigm, at the expense of the regularity of maintenance visits that occur under scheduled maintenance. Another purpose of the sustainment process as a whole, as discussed in Chapter 1 and Section 2.4, is to maximize the operational reliability of the fleet as measured by the ability to fly required missions. Figure 7 shows the

relationship between these two objectives of the sustainment process under CBM and the parameters that impact these objectives.

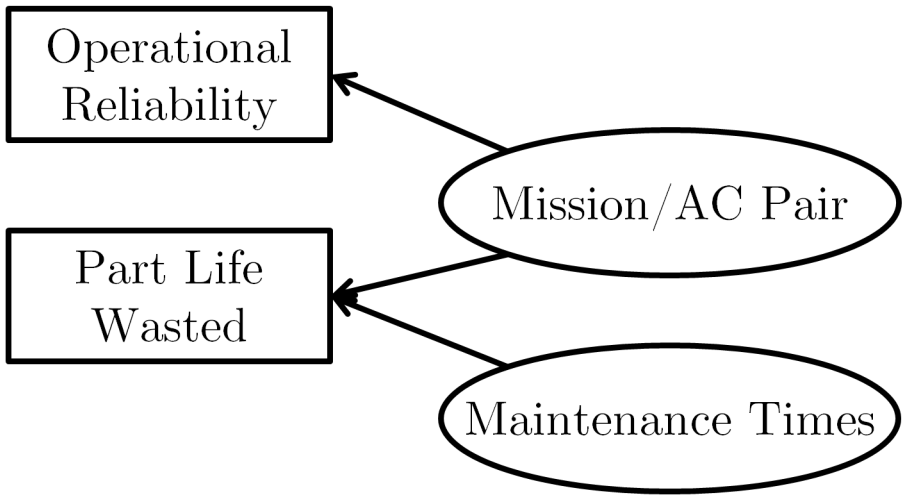


Figure 7: Condition based maintenance objectives

In Figure 7, the two objectives are effectively independent of one another. The parameters that control these objectives are the assignment of aircraft to required missions, and the scheduling of maintenance visits as failures are detected. Because the aircraft assignment is determined first, and the maintenance schedule is based on when failures are detected as missions are flown, the two objectives can be maximized and minimized, respectively, without needing to make a common decision. Figure 8 shows how this problem changes due to the desire to move to regular maintenance spacing while accounting for failure information. Under this set of objectives, the goals of sustainment become coupled.

To change the regularity of maintenance events, the maintenance schedule must be shifted by maintaining aircraft at different times. The boundaries of this decision are, on the early side, the time at which failure is detected, and on the late side, the time at which failure will occur. However, maintaining earlier than the last possible minute also increases the part life wasted from what occurred under the just in time maintenance strategy; this is the primary trade-off of the novel CBM strategy.

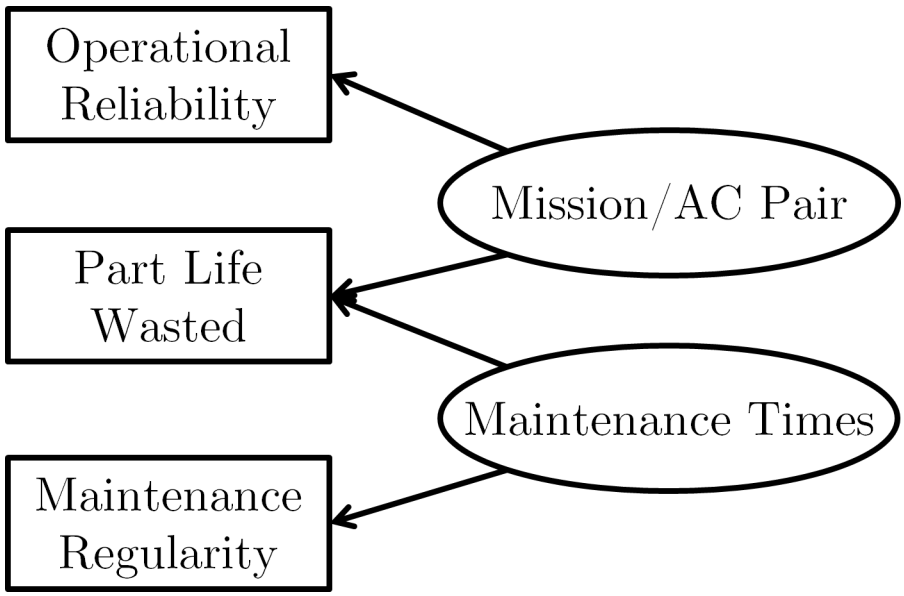


Figure 8: Condition based maintenance objectives

Maintenance times are a required variable for performing CBM differently, but these are not the only variables available. Where the aircraft assignment to missions was performed based on some default assignment logic under CBM, they present an opportunity to mitigate the additional part life wasted due to creating regular maintenance events. Depending on the total number of aircraft available to fly missions and the total number of missions required, this may even be accomplished without sacrificing any operational reliability. However, it does present a more complicated problem to solve as the two decisions (how to assign aircraft to missions and when to schedule maintenance events) couple the objectives. Furthermore, the aircraft assignment, maintenance times and the relationship between these two variables are constrained by the boundaries of the problem. The combination of objectives, decision variables and constraints suggests a solution method in the form of an optimization or decision making method; this is why the literature about these fields has been surveyed. This is also where the novel CBM maintenance paradigm gets its name: since it is a strategy for performing CBM based Mission and Maintenance Optimization of Scheduling Alternatives, it will hereafter be referred to as CBM-MiMOSA. This literature will

next be discussed, but first CBM-MiMOSA should be tied back to the literature-based maintenance paradigms discussed in Sections 2.3.1 through 2.3.3. Figure 9 shows how CBM-MiMOSA can be mapped to the high level sustainment phases of operations, maintenance, maintenance paradigms, and supply chain as was done in Figures 4 through 6.

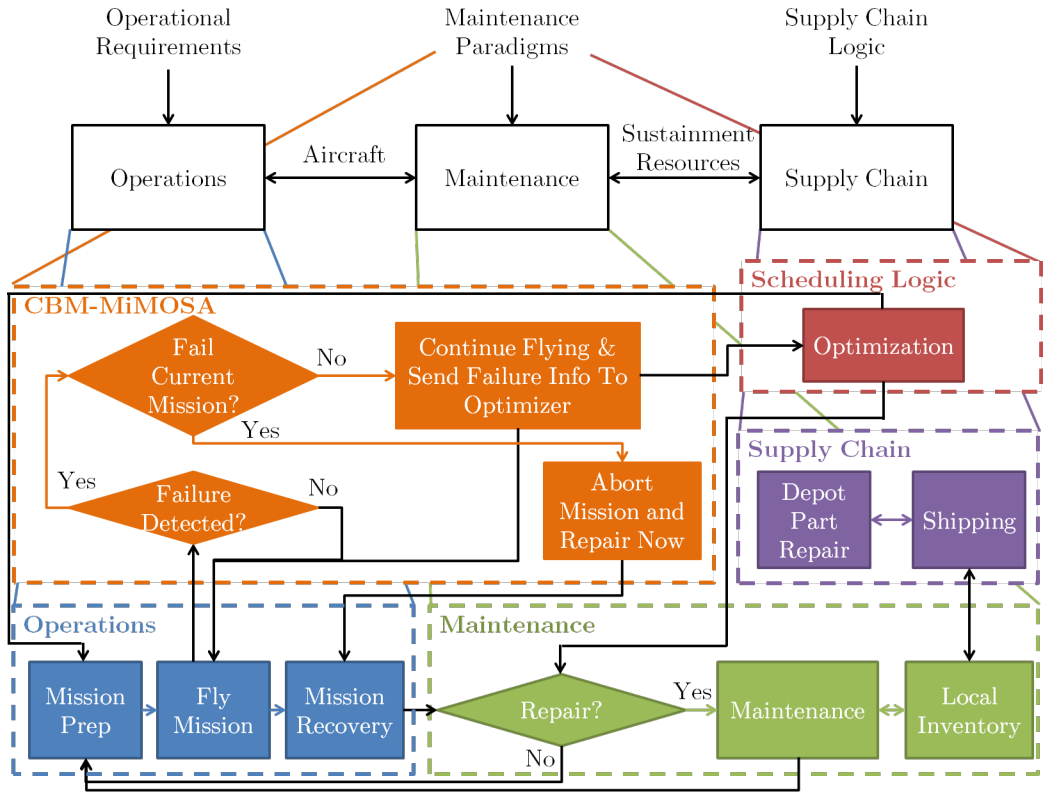


Figure 9: Sustainment steps under CBM strategy with predictable scheduling

From the field of optimization, many methods are can potentially be used. However, existing schedule optimization problems tend to use Linear Programming (LP) variants[108] and Genetic Algorithms (GAs)[24]. This is because these methods work well with the kind of variables which are available in scheduling, which tend to be integer-based, discrete choices with large numbers of constraints. As a result this literature search will limit itself to these two methods.

Linear Programming comes in several varieties[101, 13, 12, 77]. The most basic

is solved with the simplex method; integer programming is a subset of linear programming where variables can only take on integer values and Mixed Integer Linear Programming (MILP) is a hybrid between the two; quadratic programming, nonlinear programming and stochastic programming relax the linearity requirement. All these methods have been richly explored in the literature and solution methods as well as computer codes exist for solving these types of problems. LP methods return deterministic solutions, and can be completed very quickly. This last factor is ideal since whichever optimization method is used will be run many times within Sustain-ME.

GAs are based on the idea of evolution, using computer-simulated versions of mutation and breeding to improve a population of variables to find the best solution. In the past they have proved effective at solving difficult problems, and they take advantage of exploratory behavior to attempt to find global optima. Since many methods struggle at doing this, Genetic Algorithms are well-suited to problem spaces with many local extreme points[118]. However, because they find a stochastic solution each time, multiple iterations of the optimization may be needed to ensure a good solution has been found. This makes the method less than ideal for use inside a model.

Finally, decision making methods have a role in trading different solution options found by more basic optimizers. Finding the whole Pareto frontier, which contains the set of the best solutions[25], can be used for a posteriori decision making – providing all the information needed to make a decision later. Other methods, such as an Overall Evaluation Criterion (OEC) set preference information a priori in order to avoid the need for user input before proceeding. The method selected is usually based on whether the decision making inputs are qualitative or quantitative, as well as how preference information will be provided.

At the moment, there is not enough information about CBM-MiMOSA to determine whether optimization or decision making is appropriate. This topic will be

reexplored once the full approach has been developed in Chapter 3.

2.7 Modeling

In modeling, there are four main options. The first division is between mathematical models and simulation. Mathematical models look at the process as a whole and attempt to estimate its general attributes and behaviors. Many inventory base stock level assessments[7] and maintenance optimization models are of this type[10, 102, 28, 29]. They have also been used to determine when to replace parts based on PHM information to both minimize breaks and reduce cost[58, 75]. These are usually the fastest to execute and may have closed-form solutions. However, their modeling power can be limited and if time-dependent behavior is important these models are generally not used in this capacity; they are intended to rapidly provide information about what conditions result in time-dependent behavior.

If this is not sufficient, simulation models are better able to provide time-dependent information, though for stochastic processes the time behavior may change significantly between repeated model executions. Sadoun states that there are three main simulation types: Monte Carlo for stochastic processes which are invariant throughout time, continuous time models and discrete event simulations[97]. As discussed in Chapter 1, models which can capture nonstationary behavior must be constructed. For this reason Monte Carlo models should not be used. However, continuous time and discrete event simulations provide alternatives to mathematical models should these be needed. This hierarchy of modeling is presented in Figure 10.

2.7.1 Markov Chains

Markov chains are mathematical models of stochastic processes. They are fairly accurate when those processes obey the Markov property, which requires that the state of the system only be based on its previous state or a finite set of previous states, not the entire history of the system[72]. Using these models, powerful observations

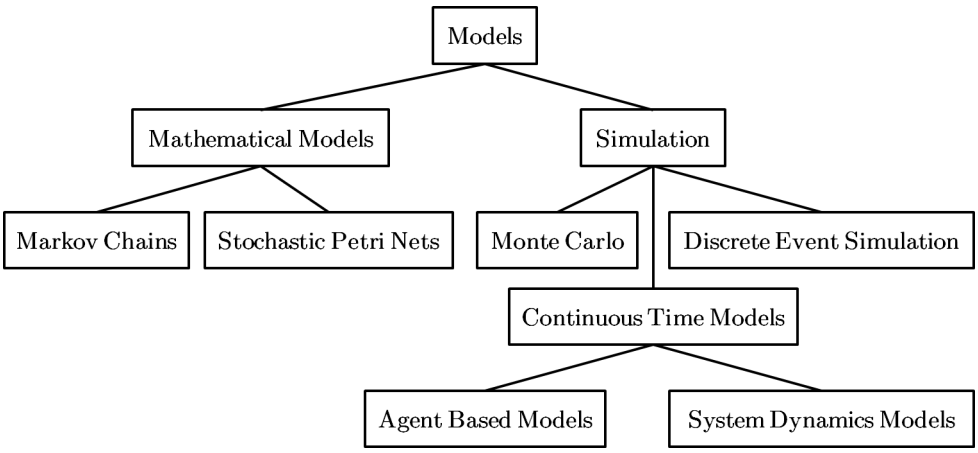


Figure 10: Hierarchy of modeling types

can be made about the long-term behavior (or lack thereof) of stochastic systems. These observations are mathematically derived from the basic rules of probability, but through linear algebra and differential equations this math can be compressed into a few representative equations which provide a closed-form solution for the long term behavior of a system.

However, while Markov chains can be developed for many stochastic problems, even complex ones, increasing model complexity limits the ability to solve for closed-form solutions. Thus, using Markov chains to solve more complex problems is often a balancing act between achieving the complexity required to capture all important effects while keeping the model simple enough to gain meaningful answers[51]. In the case of sustainment, the main difficulty of a Markov chain representation is the supply chain's rules. Though it would be possible to assume a supply chain that obeys the Markov property, such an assumption requires the supply chain to be either independent of the maintenance process's need for inventory (modeled purely as a stochastic arrival rate) or tied to inventory but without any ability to specify the time delay in receiving inventory (modeled as a finite state Markov chain). Despite the benefits of simplicity and mathematical certainty associated with Markov chains, these fidelity sacrifices in the way the supply chain would have to be modeled preclude

their use in truly understanding the sustainment process. In addition, even if these problems could be resolved, the complexity of the problem is unlikely to allow Markov chain math to be applied to find a closed-form solution.

Another problem with Markov chains relates to their main purpose: finding long term system behavior. Markov chains represent stochastic processes, and as was discussed in Section 1.3.1 stochastic processes are either stationary or nonstationary. Whether a Markov chain is stationary is determined by looking at how the probability of being in different states is distributed for a single transition event. By multiplying these probabilities together for many transition events (essentially adding up all the probabilities for getting to a given state in different ways), the probabilities for certain stochastic processes will eventually stop changing as more transition events are added. This does not indicate that the system itself stops changing, but that over many repeated cycles of the process there are set probabilities of being in any given state, regardless of the system's initial state[72]. However, not all stochastic processes display this behavior; some rotate between multiple long-term probability distributions, some are dependent on the initial state, and others never settle into a consistent behavior. Markov chains are very useful for determining when steady-state behavior will occur and what it will be, but if the stochastic process of interest is one which does not display steady-state behavior the Markov chain cannot efficiently provide insight into the process. Since this thesis has already identified a non steady-state region of the sustainment process as the region where minimal inventory and cost should occur, Markov chains have limited usefulness in identifying strategies which can help manage this region intelligently.

2.7.2 Stochastic Petri Nets

Stochastic Petri Nets (SPNs) are another attractive modeling option. They retain the ability to assess a model without needing to simulate its behavior, reducing analysis

time significantly. Unlike Markov chains they require computers to do this analysis, but this gives SPNs a greater degree of modeling power than Markov chains can achieve[51]. Despite this, SPNs use a few basic building blocks to model stochastic processes, making them simple to learn and use. Additionally, the visual nature of SPNs make them somewhat intuitive to understand and validate[124]. The model's building blocks are places, which represent states the individual elements can occupy, transitions, which link places to one another either immediately or after a timed delay, and tokens, which represent individual entities within the stochastic process. Tokens are moved between places depending on three functions: input functions require that a token be in the originating place before one is put in the destination place, inhibitor functions prevent transitions from occurring unless the inhibitor place is empty, and output functions move a token from a place to a transition. Intricate models can be constructed from these elements by building up intermediate structures such as queues, parallel activities and enabling activities, and linking these together at a higher level. This also means the pieces of the model can be tested individually to enable model verification in a more intuitive way.

Despite the wide variety of problems which can be modeled with SPNs, one important drawback of this method is that tokens are identical to one another and do not retain information about their history or specific attributes. Though colored Stochastic Petri Nets do allow for some differentiation between different *types* of entities[51], no concept of unique individuals exists in SPNs[46]. This can be problematic when implementing policy. For instance, in a maintenance process, a policy that services the individual in the queue with the easiest problem to fix could not be modeled using SPNs, even if policy were the central question of interest for the model. It also prevents metrics from being compiled at an individual level, which could be a problem, for instance, if entities with different attributes were being compared to one another.

Though to some extent these disadvantages can be managed through intelligent modeling, Stochastic Petri Nets have one more similarity to Markov chains which makes them less suitable for modeling sustainment: they also focus on the long-term behavior of the system[51]. Consequently, they are more suited to modeling sustainment than Markov chains are, but other models may be more preferred still.

2.7.3 Agent Based Modeling and Multi-Agent Systems

Agent Based Models (ABMs) and Multi-Agent Systems (MASs) are two variants of the same type of model, where agents are modeled with simple behavioral rules but their interactions lead to more complex behaviors[90]. In Agent Based Models this is known as emergent behavior[47] and is one of the main reasons these types of models are studied. According to literature, their purpose is not to solve engineering type problems but to examine natural processes[47]. Multi-Agent Systems also use the interacting agents mechanism, but are designed to find distributed solutions to complex problems[92]. In organization, they are effectively the opposite of Stochastic Petri Nets, since each individual is completely autonomous and self-aware but there is no central organization to their behavior. Though this addresses some of the problems with SPNs, it eliminates the benefit of being designed to model stochastic processes with a definite structure. As a result, MASs are also unsuited to model the sustainment process. However, it is possible that MASs would be helpful for implementing complex policies within a sustainment model, effectively filling in one of the blocks without interacting with the rest of the system.

Since ABMs and MASs are simulation models, they are more computationally intensive than Markov chains and SPNs. However, they do provide the ability to monitor how the system's behavior plays out over time, which the mathematical models cannot. The ability to see the evolution of model attributes over time is invaluable to understanding non-steady behavior, but because it will change as stochastic draws

return different values in subsequent model runs it is important to take several repetitive data points to capture the model's variability, enough that the metrics being studied (such as the mean across repetitions) stabilize. The model's size can become an issue, as large numbers of agents and repetitions may lead to long run times. Depending on the time frame for analysis, this problem may or may not constrain the choice of a model.

2.7.4 System Dynamics Modeling

System dynamics is another type of simulation model which is designed to show the behavior of complex systems over time[16]. It has the ability to model resources, the flow of entities, event delays, and feedback which means it has all the elements necessary to model a logistical process such as sustainment. System dynamics is also good at illustrating how individual processes which are simple can come together to exhibit complex nonlinear behavior. It does so by combining model elements using math from arithmetic through calculus, where the modeler defines the initial parameters needed to solve these problems. However, these models generally do not include stochasticity which limits their applicability to a process such as sustainment[107].

Aside from this flaw, system dynamics has a few other drawbacks. First, like Markov chains and Stochastic Petri Nets, system dynamics does not track entities as unique individuals[107]. Thus, like with these simpler models, system dynamics models are limited to monitoring the gross behavior of the system. This is often adequate, but depending on the information needed from the simulation more may be required. Second, system dynamics is designed for monitoring systems which evolve continuously with time[107]. Sustainment does not. It experiences distinct changes in state, though these changes occur at random times as defined by stochastic distributions. Though system dynamics is capable of modeling discrete changes, it does not do so in the most efficient manner possible. This brings the discussion to

the topic of Discrete Event Simulation, discussed in Section 2.7.5.

2.7.5 Discrete Event Simulation

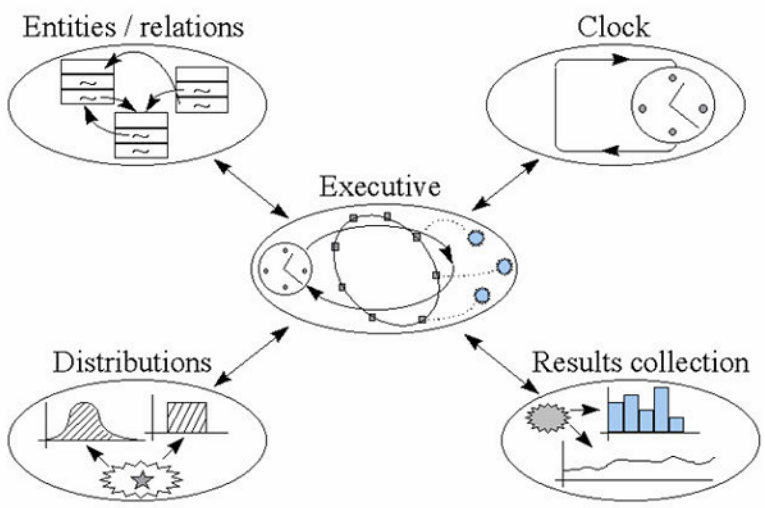


Figure 11: Discrete event simulation[103]

Discrete Event Simulation is an appropriate simulation tool for modeling the type of system described in Section 2.7.4 where the system state changes at distinct moments rather than continuously throughout time. It does so by having each element within the model compute the time when its next event will occur and compiling these into a queue of events[41]. The model then jumps from event to event, evaluating the changes that occur each time and updating the system's state. By doing so, a continuous model that could potentially require millions of time steps to simulate the timeline might only require thousands when replaced by a discrete event model. This change means a DES executes the model more efficiently, and consequently more quickly than continuous time models. Figure 11 shows a notional image of a discrete event simulation, which combines the clock, distributions associated with individual processes, the properties of unique model elements, and the data to be collected.

Because DESs are suited to logistics (and by extension the sustainment process), they are one of the most commonly used modeling techniques for logistics problems[54, 95, 107]. DES models can be tailored to the fidelity required for a given logistics

problem, which allows conclusions to be drawn about a system even when every detail of its behavior is not known a priori. This may allow for repeated modeling cycles where the concept is refined as the problem becomes better understood. Most DES packages, such as SimPy for Python, come packaged with model elements such as resources, containers and stores which can be used to build up logistics processes. The logic behind waiting for another entity to act before proceeding is also pre-implemented, meaning a process like maintenance translates fairly naturally to the DES language[81].

DES also has its drawbacks, namely the fact that events which occur at the same simulation time are still processed in the order they entered the event queue. As a result, events which should occur simultaneously actually occur in sequence. This means care must be taken in initiating the simulation's entities, or behaviors which are artifacts of the code and not true representations of the real process may emerge. The solution is usually to randomize the order in which entities are processed, but even this may just be an approximation of the true behavior. Whenever possible, logic should be applied to link the model to the real world process and use this process as the basis of programmed behavior.

2.7.6 Modeling Conclusions

Section 2.7 has described several different modeling methods from the literature and outlines their benefits and drawbacks. Table 1 was created to synthesize this information in the key criteria that will be used to determine the modeling type for Sustain-ME: whether the model type can represent stochasticity, whether it can capture time dependent behavior, and how quick the run time is. An "X" denotes that the model type satisfies the criterion, and a "/" denotes that the model type partially satisfies the criterion.

The only modeling method that can truly model stochasticity in a time dependent

Table 1: Modeling method evaluation against criteria

	Stochasticity	Captures time dependent behavior	Rapid run time
Markov chain	X	/	X
Stochastic Petri Nets	X	/	X
Monte Carlo	X		X
Agent Based Models	X	X	
System Dynamics Models		X	
Discrete Event Simulation	X	X	/

fashion and which runs fairly rapidly is Discrete Event Simulation. This modeling style is appropriate for sustainment because it captures the necessary stochastic effects, through the use of distributions on the time to complete steps; it simulates the behavior throughout time as opposed to forming high level conclusions about the steady-state behavior; and it runs fairly rapidly, gaining efficiency by scheduling events and jumping from one to another. Additionally, DES has the built-in ability to model queues for resources, another aspect of sustainment. For these reasons, a DES framework was chosen to build Sustain-ME for this thesis.

2.8 Conclusions

This chapter began by explaining several different maintenance philosophies from the literature to ground the maintenance paradigms that are modeled in Sustain-ME. Next, an enabling technology for CBM, prognostic health management, was introduced. The literature provided information about how PHM might work in the real world, but little information about how effective it is expected to be. For the purposes of this thesis, the overall effectiveness is a much more important quality for the PHM, and therefore the PHM will be modeled at this level of fidelity. Next this chapter explored the different options for creating models; discrete event simulation was determined to be the most appropriate modeling method for the sustainment process. At this point the sustainment process was described in greater depth, highlighting the differences between the different maintenance paradigms that will be modeled in

this thesis. Next maintenance metrics were defined, and supply chain management was briefly discussed to provide a foundation for a portion of Sustain-ME that will recieve less focus in this thesis. Finally optimization and decision making methods were explored to help inform the decision about how to develop for the purpose of demonstrating the abilities of Sustain-ME. Having now explored the concepts relevant to the creation of a sustainment model, Chapter 3 will next develop a framework for developing and testing both Sustain-ME as well as the different maintenance paradims that will be compared with it.

CHAPTER III

MODEL FORMULATION AND EXPERIMENTS

In Chapter 1 a set of paradigm shifts for military aircraft sustainment were introduced and a new modeling environment able to predict the effects of those paradigms, Sustain-ME, was introduced. In Chapter 2 different fields and methods from literature were introduced to provide the background necessary for implementing Sustain-ME and its example use case. This chapter focuses on synthesizing the two to formulate the details of Sustain-ME and develop the experiments that will be used to check that it is modeling the right effects.

Keep in mind that the purpose of Sustain-ME is not to capture all sustainment behaviors observed in the real world, but to create a basis for making decisions about sustainment into the future. Thus Sustain-ME should capture enough aspects of sustainment that general interactions between different sustainment processes can be observed, while future trade-off studies will define and incorporate any relevant aspects of new sustainment strategies that are tested. For instance, this thesis will demonstrate Sustain-ME by comparing different maintenance paradigms; future studies might focus on supply chain decisions and in that case the modeled supply chain would be developed to reflect those decisions. However, this does not mean the supply chain will be neglected in this thesis. Since the most relevant aspect of the supply chain is that it is based on limited resources with long turnaround times for refurbished parts, modeling this primary behavior should provide a reasonable approximation of real world behavior without needing to explore in great detail the different options the supply chain at a high level of fidelity.

The first part of this chapter will focus on the behaviors that must be captured

in Sustain-ME when the focus is maintenance. In doing so, Research Question 2 will be explored and answered. Once Sustainme has been conceptually developed, further questions will be explored. Hypotheses 1 and 2 will be revisited, and experiments to test them will be described based on the specifics of Sustain-ME. Finally, CBM-MiMOSA will be revisited to develop the mathematical basis for this strategy.

3.1 Sustainment Modeling

Research Question 2 asks, "What level of fidelity is required to capture the major trends within sustainment?" Because this question is broad, a specific hypothesis could not be developed to determine an answer. Instead, the characterization of sustainment begun in Chapter 2 will be continued, drawing from literature to provide specific steps within the sustainment process and distributions associated with these steps. These steps will form the basis for Sustain-ME's logic. In specifying the exact behavior of the sustainment process modeled here, the thesis's stated goal of providing a transparent, open-source sustainment modeling environment will be partially achieved. From the basis of this formulation, any future efforts will have the ability to examine the currently modeled behavior and either deem it appropriate or update it to match future problems. The completion of the goal will come through demonstrating Sustain-ME's adherence to these intended behaviors, and demonstrating the types of studies that such a model enables. This section addresses the behaviors of Sustain-ME derived from literature. After developing the main behaviors, additional assumptions that had to be made to create Sustain-ME and their justification are discussed.

3.1.1 Operations Modeling

Figures 4 through 6 list the same three steps for operations: mission prep, fly mission, and mission recovery. Thus Sustain-ME's operational modules will remain largely the same through the different maintenance paradigms of the example use case. Faas and

Iakovidis describe the following steps within the mission preparation phase: mission scheduling, preflight inspection, refueling, load weapons, engine start, final systems check, taxi, and takeoff[39, 56]. These steps are supported by several entities and resources: flight chief(s)[110], crew chief(s)[39, 109], ground crew(s)[39], and runway(s). Depending on the step, different resources or entities may be required to be present for completion of the step; for instance, the flight chief is required to be present for mission scheduling. These limitations placed on Sustain-ME represent real world military regulations, and can easily be updated as regulations change. Since these entities and resources are limited in number, they can be represented in Sustain-ME by the resource class, which creates queues whenever resources are currently in use. This preserves the real world behavior associated with work flows for individuals. Additional limitations imposed by the real world include the fact that the ground crew must be present for preflight inspection through taxi, the crew chief must be present if preflight failure inspection is performed (for the maintenance paradigms modeled in this thesis, no such check is required, meaning that the crew chief's role is merely as a placeholder), and the runway must be available for takeoff. Incorporating these limited resources helps to capture the true amount of stochasticity associated with mission preparations, as merely placing delays on each step without requiring limited resources would create a more predictable system than is realistic. Finally, incorporating limited resources into Sustain-ME provides the opportunity to perform trades on the effect and cost of different resources for improving sustainment metrics. In this thesis, the decision will be made with a calibration activity described in Chapter 4.

Figure 12 illustrates a more in-depth view of mission preparations for a military aircraft fleet as derived from literature. It shows the relationship between different mission prep activities and the resources required to carry them out. Activities are defined by distributions and represented as rectangles, while waits for different resources are defined by queues and represented as ovals. Figure 12 represents the

steps that a *single aircraft* must take to complete mission prep. This distinction is important because the fleet is composed of many aircraft completing the same steps in parallel; thus the actions of one aircraft can influence another through the mutual need for limited resources. Also, it bears stating that Sustain-ME observes the effects of individual aircraft operating simultaneously, not the behavior of the fleet as a whole.

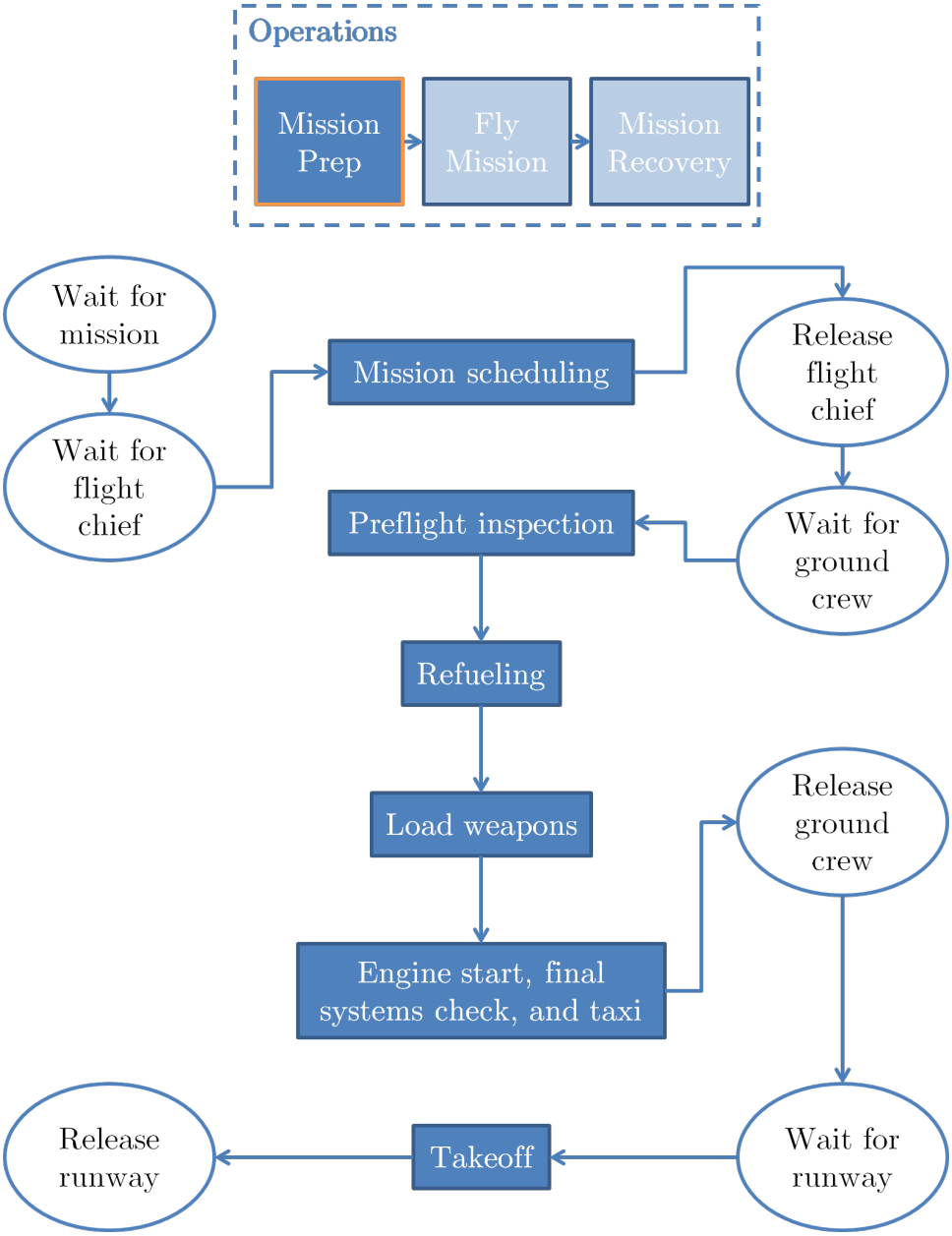


Figure 12: Mission preparation activities

The mission scheduling step is fairly straightforward; during it pilots and aircraft receive orders about a mission they will fly. The preflight inspection step checks for any failures that might have been previously missed. Due to the way failures are modeled in this thesis, only flight hours count towards part wear. This means that, under reactive maintenance all failures occur while on missions, and under CBM with PHM, failures will either occur while on missions or will be predicted and prevented. As a result, the delay for preflight check is still carried out, but no failures are ever found. This represents another assumption of Sustain-ME that could be adjusted depending on the real world process being modeled. The refueling and load weapons steps are self-explanatory. The engine start, final systems check and taxi step is also self-explanatory, factoring in the same logic by which the preflight inspection does not discover failures. Finally, the takeoff step is again self-explanatory.

Table 2 details the time distributions associated with the mission prep steps. These time distributions represent the fact that there is uncertainty associated with how long different activities will actually take to carry out. They are also derived from the literature, and are estimates of the distributions associated with the real world equivalents of these processes. These distributions are one of the simplest model parameters to change, especially if the parameters of the distribution change but not the distribution shape. If the distribution type changes (i.e. from triangular to normal) this represents a barely more complicated change. This is one of the ways Sustain-ME can be updated to represent current sustainment processes. Most of the distribution of step durations in this thesis are based on the triangular distribution, for which the probability density function (PDF) is presented in Equation 3.

Table 2: Mission preparation time distributions

Step	**Duration Distribution**
Mission scheduling	Tri(30,45,60) min
Preflight inspection	Tri(50,60,70) min[39]
Refueling	Tri(20,22,25) min[56]
Load weapons	Tri(45,60,75) min[56]
Engine start, final systems check, and taxi	Tri(7,10,12) min[39]
Takeoff	Tri(2,3,4) min[39, 56]

$$PDF_{Triangular} = \begin{cases} \frac{2(x-a)}{(c-a)(b-a)} & x \in [a,b) \\ \frac{2}{c-a} & x = b \\ \frac{2(c-x)}{(c-a)(c-b)} & x \in (b,c] \\ 0 & else \end{cases} \tag{3}$$

For the fly mission phase of operations, the only step is to fly the mission, which according to Iakovidis follows a truncated normal distribution with mean of 1.3 hours[56]. However, Faas used a truncated normal distribution with mean of 2 hours[39]. Since these two sources did not agree, their mean values were used to bound the distribution used in this model, which was chosen as a triangular distribution for consistency and boundedness. Again, the behavior of Sustain-ME matters more than the specific input and assumption values, assuming that those values are reasonable, because the goal of the thesis is to provide a framework and ensure that relevant effects are captured. Thus the fly mission duration is modeled as a triangular distribution with minimum of 1.3 hours, maximum of 2 hours, and mode of 1.5 hours. The mode was chosen to mimic the behavior of a lower truncated normal distribution, which will be positively skewed (i.e. have a longer tail toward high values).

For the mission recovery phase of operations, Faas cites the following steps: landing, and parking & recovery[39]. The distributions for these steps are Tri(14,15,16) minutes and Tri(5,7,9) minutes respectively. Additionally, there is a servicing step

that counts as downtime in the A_O computation, as do the rest of the steps of maintenance. However, because it does not require maintenance resources and occurs after every flight, regardless of whether failure has occurred, it is encompassed in the operations portion of sustainment – this can be seen as the operational component of maintenance and it includes actions such as checking and replenishing fluids. The distribution associated with servicing is Tri(45,60,75) minutes[39]. As with mission preparation, certain resources are required to perform these activities. The runway is required to perform landing, and the ground crew is required to perform park & recovery. However, the ground crew is also required for subsequent steps which fall under the maintenance umbrella, so they are not released immediately after parking & recovery have been completed. Figure 13 shows the activities associated with mission recovery using the same format as Figure 12.

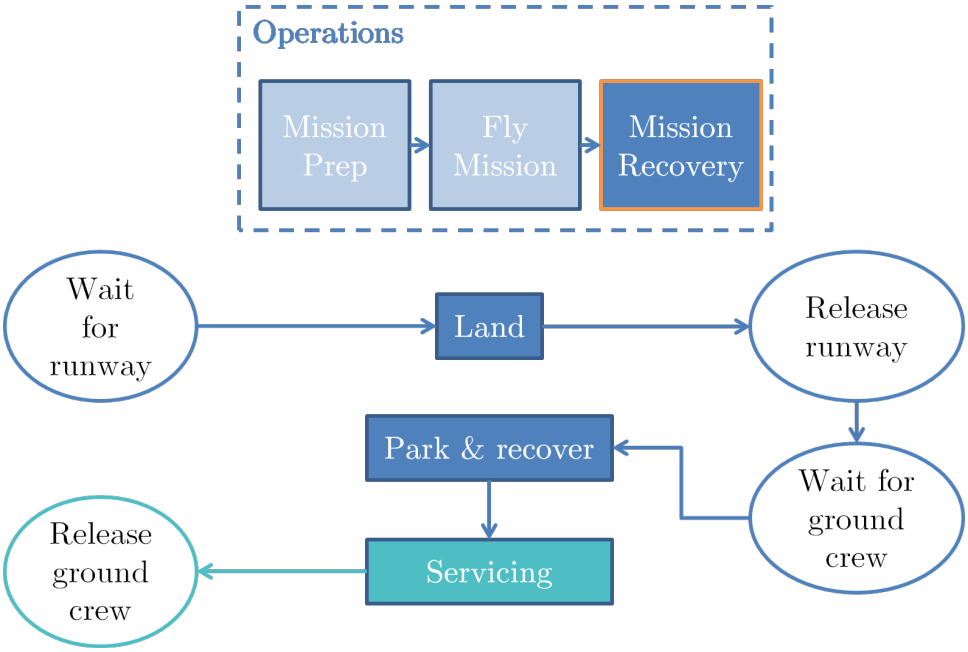


Figure 13: Mission recovery activities

3.1.2 Maintenance Modeling

Figures 4 and 6 list almost the same three steps for maintenance: first a decision point for whether the mission was aborted (reactive maintenance) or whether repair should

occur (CBM paradigms), next a maintenance phase and final a local inventory step. Thus Sustain-ME's maintenance modules will also remain largely the same through the different maintenance paradigms of the example use case. It should be noted that Figure 5 had slightly different phases within the overall maintenance process, but that this paradigm will not be modeled using Sustain-ME.

Faas lists the following steps within the maintenance phase: repair need check, document corrective actions, remove LRU, wait for part to arrive from local inventory, write inventory paperwork, and replace LRU[39]. The individual rules for what passes a repair need check vary from paradigm to paradigm, and this has been covered in Figures 4 through 6. The resources required to carry out maintenance activities are the maintenance staff and maintenance facilities[123]. Think of the maintenance facility resource as a berth within a building rather than the building itself. The detailed breakdown of maintenance steps is shown in Figure 14.

The repair check logic is, as discussed, dependent on the specific maintenance paradigm being modeled. The document corrective actions step involves writing a report on the specific parts that failed and the actions planned to fix the failure. The remove LRU step involves taking the broken part off the aircraft, and the send parts to depot step involves shipping the part away to the vendor to be refurbished. The local parts available check looks for replacement parts in local inventory and preferentially uses these if they are available. If they are, a wait for them to be taken out of inventory occurs. If they are not available in local inventory, the aircraft must wait until replacements arrive from the depot. During this time the maintenance staff are released because they are not needed while the aircraft is awaiting parts; the maintenance facility is not released because the aircraft must remain in place during maintenance. Once the parts arrive from the depot, they are then sent from inventory and after this point the two versions of the maintenance behavior converge. Once parts are on hand and maintenance staff have been reacquired (if they were initially

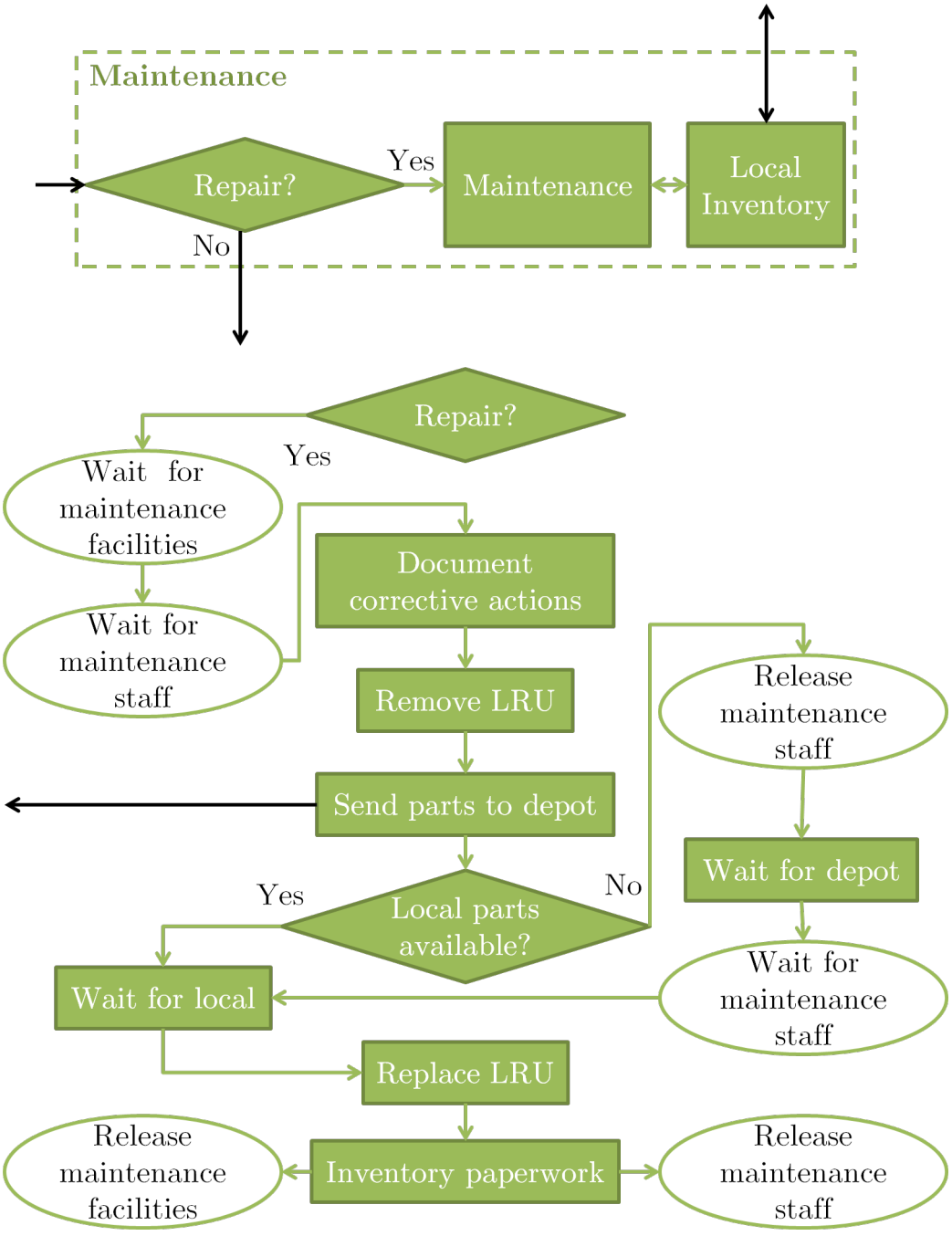


Figure 14: Maintenance activities

released), the new part is installed on the aircraft. Next the inventory paperwork step requires that paperwork related to the spare parts on hand and on order be completed, after which the maintenance staff and facilities are released.

Table 3 contains the distributions associated with the activites in Figure 14.

Table 3: Maintenance time distributions[39]

Step	**Duration Distribution**
Document corrective actions	Tri(5,10,15) min
Remove LRU	Tri(45,60,70) min
Wait for local	Tri(0.5,2,2.5) min
Replace LRU	Tri(60,84,120) min
Inventory Paperwork	Tri(5,10,15) min

3.1.3 Supply Chain Modeling

Figures 4 through 6 list the same two steps for the supply chain: depot part repair and shipping. Thus Sustain-ME's supply chain modules will remain the same through the different maintenance paradigms of the example use case. Figure 15 shows the steps of the supply chain process, which largely resemble the higher level phases of the supply chain; these steps are self-explanatory. The distributions associated with shipping to the depot, depot refurbishment, and shipping to the base are U(0.25,0.5) days[39], Tri(69,87,104) days[114, 65], Tri(0.1,0.3,0.5) days[39] respectively. Equation 4 shows the PDF for a uniform distribution.

$$PDF_{Uniform} = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & else \end{cases} \tag{4}$$

3.1.4 Additional Modeling Assumptions

Though Sections 3.1.1 through 3.1.3 have discussed the steps of sustainment and how long these are expected to take, as well as the resources required to carry out these steps, several aspects of sustainment still remain to be determined before a model

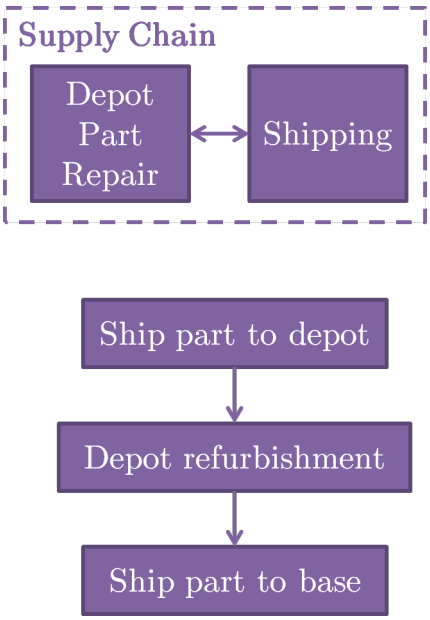


Figure 15: Supply chain activities

can be created. First, the specifics of the PHM as implemented in Sustain-ME must be determined. Chapter 2 described how PHM has been modeled in the literature and briefly conceptualized how it would be addressed in this thesis, but this section will present a formal explanation of the concept. Next, the issue of how often aircraft can fly missions is addressed, followed by the replacement part ordering strategy that will be employed in Sustain-ME as part of the supply chain logic. Next the level of complexity for aircraft components that must be modeled and the reliability of those components is discussed. Finally the calibration of Sustain-ME's resource levels is discussed.

First, the determination of how to model PHM will be completed. Due to the precedent set in using the surrogate measure of detection time as a percentage of part life to represent the PHM[78, 121], this thesis will model the PHM using distributions around the detection time based on Malley's findings. The detection time distributions will be used to calculate the percentage of a modeled component's life at which failure is detected, as shown in Figures 16 and 17. Once the failure is detected it is assumed that the simulation perfectly predicts when failure will occur.

This second assumption mirrors Malley's implementation. Though this assumption is slightly unrealistic, it isolates the effectiveness of the maintenance paradigm from the effectiveness of the PHM. Also, the maintenance paradigm will operate the same whether false alarms are included or not because maintainers can only rely on the information at their disposal. The internal functioning of the PHM model itself is not relevant to the manner in which the maintenance logic determines when to schedule maintenance; on the contrary, only the outputs of the PHM model (a failure detection time and predicted failure time) matter. Other methods which assume a PHM with detection lead times always sufficient to order parts in advance [39, 123, 56] will not be adequate to answer the questions that will be asked in this thesis. In fact, the detection lead time may be a significant factor in comparing the effectiveness of different maintenance paradigms. When lead times are sufficiently long the PHM system is expected to perform better, and when they are short it is expected to be less effective.

To capture the essence of the PHM without modeling all the details, the time at which the PHM will detect failure for any individual part is illustrated in Figures 16 and 17. Figure 16 shows how failure will be modeled. When a new part is installed on the aircraft, a failure time for that part will be generated from a random distribution. As stated in Chapter 2, the exponential distribution will be used to represent component reliability because it will be assumed that Sustain-ME represents the bulk of the aircraft's life, rather than the burn-in and wear-out periods that characterize the early and late life of the system. The reliability will be based on values derived later in this section, but importantly, will be based on the wear and tear on the part accrued during flight, rather than the total clock time. Once the failure time is drawn from the distribution, the part "knows" when it will fail, and every flight hour flown on the part deducts from the time remaining until failure. For the maintenance paradigm where parts are used until they fail, this will continue until the part fails,

at which time the rest of the simulation will become aware that the part needs to be fixed. For the two paradigms with a PHM implemented, the simulation is able to become aware that the part has an impending failure only once a detection event has occurred. Figure 16 shows the failure time being drawn from the exponential distribution.

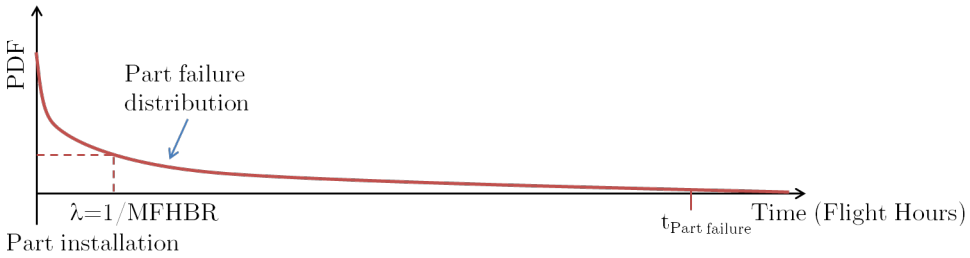


Figure 16: Illustration of Sustain-ME part failure

The detection event is illustrated in Figure 17. Recall that any individual part within Sustain-ME will acquire a failure time when it is first installed on the aircraft. At the same time, a detection time will be generated based on the specific value drawn from the failure distribution. The detection time is also based on a distribution to simulate the fact that the warning time for a part is not likely to be deterministic. The distribution in this case will be a truncated normal distribution centered around some value μ and truncated between 0 and the failure time. μ will be computed as a percentage of the specific failure time for the part at the time of installation, for instance at 75% of $t_{Partfailure}$. This percentage is the value that will be varied as the PHM detection lead time to simulate different levels of effectiveness for the PHM. Once both the failure time and distribution time are known, both will be deducted for each mission flown until the detection time is reached, at which point the rest of the simulation will gain full knowledge of the failure time. Table 4 lays this process out in three steps.

The next discussion of a modeling decision involves the number of missions that may be flown by any given aircraft over the course of a single day. Multiple sources

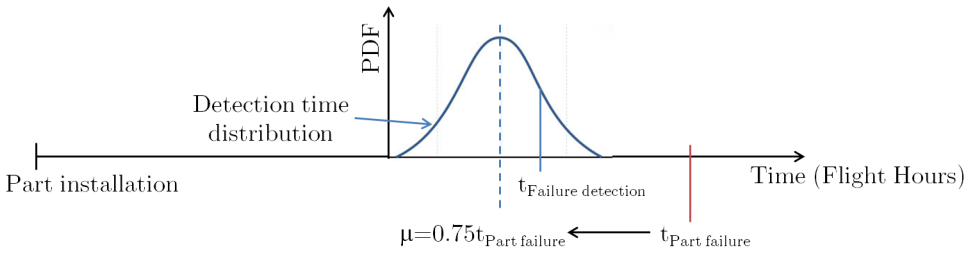


Figure 17: Illustration of Sustain-ME part failure detection

Table 4: Steps required to generate specific part failure and detection times

Step 1	Draw an instance of failure time from failure distribution $t_{Partfailure}$
Step 2	Determine mean of detection time distribution for this part instance $\mu = Detectionleadtime * t_{Partfailure}$
Step 3	Draw from the detection time distribution with mean μ $t_{Failuredetection}$

reference the idea of quickly turning around to fly secondary missions immediately after completing a primary mission [39, 56]. However, the guidelines for doing so are at the discretion of individual fleets; these secondary missions occur only as needed and the limit on how many may be flown by a single aircraft or pilot in a day will depend on the mission tempo. During heavy duty operations this number may be higher to allow the Air Force to achieve target requirements, whereas during peacetime the number of missions may be limited to one per day, or one every few days. In reality, even the time of year may affect flight scheduling [56], though this thesis will use regular flight scheduling over time to show that, even with the best case scenario of perfectly even flight scheduling, nonstationary behavior still appears. For the purpose of this model, turn-around missions will not be modeled. However, this assumption could easily be adjusted if a specific scenario called for it.

Another modeling decision that must be made and justified is the inventory ordering strategy, or the logic by which replacement parts are ordered to replenish the inventory. The strategy that is appropriate will depend on the scenario being modeled; in many cases, storage of inventory and shipping costs are primary drivers and

these suggest much different scenarios than when these effects are not accounted for. Since the overall goal of this thesis is to develop a model that can be used to model a variety of strategies, the formulation of Sustain-ME does not prevent modeling this type of problem. However, for the maintenance strategies modeled to demonstrate Sustain-ME, the issue of storage space and shipping costs are tangent to the problem of interest. A better approach is to use a one-in-one-out ordering strategy that has been used before in sustainment models, such as the one Faas created[39]. Muckstadt refers to this strategy as an (s,s-1) ordering strategy and proves that under certain conditions, this strategy is an optimal one because it matches ordering directly to demand[85]. One other advantage of this strategy is that inventory will be directly replaced, making it easy to identify the inventory level needed to achieve desired performance.

The final modeling decision concerns the required level of complexity needed to model the aircraft's individual components in implementing the failure portion of the simulation. In reality military aircraft have thousands of parts and modeling them all is infeasible due to both computing power and the availability of data. However, statistical methods can be used to mathematically represent many components with a few representative ones[98]. This suggests that, even if data cannot be found for the reliability of all an aircraft's components, using representative parts is not as problematic as it might seem. The manner of choosing which representative parts to model should be based on a few important parameters. First, parts that have high reliability are expected to have a different effect on system behavior than parts that have low reliability. Care should be taken that parts with low reliability are not chosen with reliability that is *too* low, since for the aggregate part the possibility exists of having multiple failures on the same mission, which complicates the modeling significantly. On another axis, the cost of parts could impact which maintenance paradigms are best from a financial perspective. Recall that an overall metric of the

system is inventory cost, and for some parts the cost alone might not justify paradigms that require early replacement such as what occurs under CBM or will occur more strongly for CBM-MiMOSA. A third axis of repair time or depot turn around time could also have an impact on the effectiveness of maintenance optimization, but this effect will be reserved for later discussion if the results of maintenance optimization warrant it. For this reason only reliability and cost will be included in helping to determine representative parts for Sustain-ME. Combining reliability at low (but not too low) and high values with cost at low and high values gives six parts that are necessary to be modeled.

In order to determine realistic values for overall aircraft reliability, the literature was once again examined. Though any military aircraft could be used to model military aircraft sustainment (the F-16 would be a good option because of the wealth of data available in the literature), one in particular stands out as an appropriate choice. The F-35 Joint Strike Fighter (JSF) is a new system being built and planned for at the same time as the paradigm shift is occurring in the way the Air Force will sustain aircraft. Due to this fact it will, as mentioned in Chapter 2, be equipped with an automated PHM system at the fleet level (ALIS). Thus the F-35 is a good aircraft for which to demonstrate the relative merits of different current maintenance paradigms. The predicted reliability for the JSF was found to be 6 Mean Flight Hours Between Failure (MFHBF)[116]. Since six parts with two distinct reliability values are to be modeled, these were chosen so that the overall aircraft reliability would be 6 MFHBF and so that the two values would be distinct. This holds true for part reliabilities of 25 MFHBF, 25 MFHBF and 300 MFHBF. This determination was based on the standard assumption of exponential part failures, which holds true over the middle part of the bathub curve discussed in Section 2.1.6. For exponential failures the individual part means aggregate as shown in Equation 5.

$$MFHBF_{All} = \frac{1}{\sum_i \frac{1}{MFHBF_i}} \tag{5a}$$

$$MFHBF_{All} = \frac{1}{\frac{2}{MFHBF_{High}} + \frac{4}{MFHBF_{Low}}} \tag{5b}$$

$$6 = \frac{1}{\frac{4}{25} + \frac{2}{300}} \tag{5c}$$

Since cost is only incorporated into Sustain-ME when differentiating parts, part costs may be relative. To keep cost and reliability values consistent, the cost values will have the same ratio as the part reliabilities (1 : 12). For simplicity, costs of 1 unit and 12 units respectively will be used.

The final aspect that should be addressed is the determination of what resource levels, other than inventory, should be used for Sustain-ME. In reality, these decisions are made by studying the budget and determining the most cost effective way to meet operational requirements. Models may be created to determine staffing levels, but more likely these decisions will be made gradually as they are needed. To emulate this process with Sustain-ME, a resource level that creates adequate performance but does not move past the point of diminishing returns will be selected. Though in the real world this would be done for each maintenance paradigm independently, for a modeling environment that is intended to compare different options on a similar foundation the best approach is to select one level and use it for all comparisons. If individual adjustments need to be made, these will be identified and discussed as part of the study. The metrics used to determine the appropriate resource levels should be A_O and R_O, as is true for the rest of the studies in this thesis. However, to facilitate decision making several aggregate values such as the mean, standard deviation, maximum and minimum of the response over time will be collected and used as surrogates for the full response. This will allow the effect of several resources to be examined rapidly and an overall combination of resources that is best for the

fleet to be chosen.

3.1.5 Modeling Conclusions

Research Question 2 asked what level of fidelity is required to capture the effects of military aircraft sustainment so that decision making can be facilitated. Section 3.1.1 through 3.1.4 have addressed this question by searching the literature for descriptions of military aircraft sustainment processes and compiling both activity flows as well as distributions associated with the time to complete different activities. In the few cases where information was not available in literature, engineering judgement and experience was applied. Additional aspects of sustainment were also explored in the literature and used to determine how to model the PHM, what rules to apply to aircraft assignment to missions, how to order parts, and how to model the aircraft's reliability. The activity flows, distributions and rules listed in this section will form the basis of the logic modeled within the sustainment trade-off environment. However, part of the benefit of creating Sustain-ME based on a clearly specified set of rules is that it should be easy to update under different sets of assumptions or with a different research focus. Appendix A, which provides all code used to build Sustain-ME, will also be a helpful reference for recreating or updating this work.

3.2 Hypotheses and Experiments

Chapter 1 introduced two hypotheses for testing the formulation of Sustain-ME. Now that more details about Sustain-ME are known, the specifics of how these hypotheses will be tested using this modeling environment can be discussed. Hypotheses 1 and 2 are reproduced in Sections 3.2.1 and 3.2.2 so that they can be revisited and experiments to test them can be developed.

3.2.1 Hypothesis 1 Testing

Hypothesis 1: The relationship between A_O and R_O is complex and cannot be represented by a simple correlation.

Hypothesis 1 was posed because the two metrics measure similar aspects of sustainment, but measuring both is not required from a military contracting context. However, it is predicted that there will be situations for which A_O and R_O will demonstrate different trends, suggesting that using only one metric might make the sustainment process appear to be performing well when in fact it is suffering in another metric dimension. This is predicted in part due to the fact that the availability metric, A_O, should not be maximized under the best circumstances. As mentioned before, A_O must be less than 100% for a fleet that is correctly operating, because it is unrealistic to expect never to have to perform maintenance[1]. There may be some cases where the availability is high, making sustainment appear to be performing well, when in fact aicraft are available because they are not flying and are simply waiting in one of the states that counts as available. If this is the case, the R_O would most likely be low to reflect what is occurring.

Since all that is needed to support Hypothesis 1 is an example of a reversal in trends between A_O and R_O for different sustainment assumptions, two scenarios will be posed here that are expected to demonstrate different A_O to R_O relationships. These will then be tested with Sustain-ME to determine if they in fact lead to different correlations between A_O and R_O, and if this is the case Hypothesis 1 will be supported.

The two scenarios chosen are based on the initial inventory provided to the fleet, and the resources provided to sustainment in the form of flight chief, crew chief, ground crew, runway, maintenance facilities, and maintenance staff. It is expected

[1]Even the zero maintenance paradigm acknowledges this[5]

that, as the inventory is increased, both A_O and R_O will increase because the additional spares will reduce the average time that aircraft are required to wait for repairs. This should increase A_O by increasing the proportion of uptime versus downtime. It should increase R_O due to the greater number of aircraft available on average, as these aircraft are no longer unavailable due to a wait time in maintenance. On the other hand, as sustainment resources associated with the mission prep activities are decreased, it is expected that though R_O will decrease, A_O will increase. This is because significant queues for these resources could create delays for aircraft waiting to be sent on missions, leading to a drop in R_O if aircraft are not able to fly these missions in a timely fashion. However, A_O would still remain high for this scenario, because the wait time would take place in a state that is considered as uptime – it contributes positively to A_O. Furthermore, if the R_O is affected, the fleet's aircraft will have less total flying time and therefore fewer overall failures, leading to less time spent in maintenance as well.

The test for Hypothesis 1 will vary the initial inventory and resource levels independently. To avoid conflating two trends, reasonable inventory values will be used when studying the effects of resource levels and reasonable resource levels will be used when studying the effect of inventory. Both will be varied and the effect of each variable on A_O and R_O will be examined to determine if trend reversal occurs. The correlation between A_O and R_O will also be plotted for these cases.

3.2.2 Hypothesis 2 Testing

Hypothesis 2: At the conditions cited in the Air Force sustainment paradigm shift, where minimal inventory is selected to meet a target value of 70% A_O, and where maintenance is performed based on a condition based maintenance policy, stochasticity will dominate the performance of sustainment.

Hypothesis 2 is simple to test because it aligns with model distinctions that are already being planned. Sustain-ME with the CBM maintenance paradigm is expected to exhibit stochastic behavior at inventory levels that provide approximately 70% A_O. Testing this simply requires that this version of Sustain-ME be run for a variety of initial inventory investment levels and that the region around an average of 70% A_O be characterized as stationary or nonstationary. To do so, not only will the operational availability be examined but the operational reliability (percent of missions flown) as well. The behavior of these two metrics over time will be highly variable and will not show a clear trend if the Hypothesis is supported by testing. Also indicative of the stochasticity at play, the results on subsequent repetitions of the same model (analagous to running experiments for different but similar fleets or the same fleet in different years) should exhibit nonstationary behavior that is different from what was observed for other repetitions. If these are true, then Hypothesis 2 will be supported; if clear time-dependent trends emerge for this inventory level, the hypothesis will be falsified.

3.3 CBM-MiMOSA Strategy

Chapter 2 described the idea behind the CBM-MiMOSA, a novel maintenance strategy that will use the assignment of aircraft to missions and mainenance visit scheduling to try to create more regular maintenance visits, in turn ideally reducing the inherent nonstationary of CBM under a limited inventory situation. Having described the parameters which are available to an optimizer, the actual problem to be optimized must now be defined mathematically. Then the optimizer that will be used will be chosen from the options discussed in Section 2.6.

3.3.1 Problem Definition

CBM-MiMOSA will use an optimizer to determine the ideal time to schedule maintenance for aircraft as well as the missions to assign to individual aircraft. To do

so, it must have access to the fleet's PHM data and current inventory levels in order to perform maintenance scheduling as well as access to mission requirements and aircraft flight history to perform mission assignments. The objectives of this problem are to create a steady flow of aircraft to maintenance while still using as much part life as possible, and while continuing to fly as many of the required missions as possible. The formulation as an optimization problem is meant to translate the high level objectives into actionable information about which aircraft should be flown and maintained and at what times. Therefore the different goals and limitations of the problem must be translated into mathematical form, and these must be directly related to the degrees of freedom available within the simulation, namely those related to aircraft operations.

The first of these goals, as stated above, is to create a steady stream of aircraft into maintenance at regular intervals. However, the bounds of the problem matter as well. Over the period of operations for the fleet, the goal is to have aircraft visit maintenance evenly but also for them to fill the timeline completely; in other words, if the number of maintenance visits over this period is n, and the time period of operations is T_{end}, the time between each pair of temporally adjacent maintenance visits will ideally be T_{end}/n. At this point it is helpful to define a set of variables T_1 to T_n representing the time at which n subsequent visits to the maintenance process occur. Thus the difference between each pair of maintenance visits, T_{i+1} - T_i, should be as close as possible to T_{end}/n. This is formalized mathematically in Equation 6.

$$\begin{aligned} \text{given} \quad & T_{end} \text{ (time period of interest)} \\ & n \text{ (number of maintenance visits in } T_{end}) \\ & T_i \text{ (time of } i^{th} \text{ maintenance visit)} \\ \text{target} \quad & T_{i+1} - T_i = \frac{T_{end}}{n} \end{aligned} \tag{6}$$

When the number of maintenance visits and the time period are defined as the entire span of operations for the fleet, the value T_{end}/n is also equivalent to the mean time between repairs for the fleet, μ_F. Thus the objective function representing steadiness for maintenance visits can be formalized for any two adjacent maintenance visits as minimize $|T_{i+1} - T_i - \mu_F|$. Since the absolute value function dictates nonnegativity, the overall objective function can be created by summing the individual objective function over all pairs of adjacent maintenance visits without worrying about long intervals and short intervals canceling. Also, presuming some initial conditions for the problem, there may have been another maintenance visit prior to the optimization being run. To keep continuity between the optimization period and previous operations, the most recent prior maintenance visit can be integrated into the objective function as well, where the difference between the first maintenance visit during optimization and the most recent prior maintenance visit should also equal μ_F. This objective function is shown in Equation 7 as an update of Equation 6.

$$\begin{aligned} \text{given} \quad & T_{end} \text{ (time period of interest)} \\ & n \text{ (number of maintenance visits in } T_{end}) \\ & T_i \text{ (time of } i^{th} \text{ maintenance visit)} \\ & \mu_F = \frac{T_{end}}{n} \text{ (mean time between failures for the fleet)} \\ \text{minimize} \quad & \sum_{i=0}^{n} |T_{i+1} - T_i - \mu_F| \end{aligned} \tag{7}$$

Having quantified a function for defining a steady flow of aircraft into maintenance, a function for quantifying part life wasted must now be defined. As stated in Section 3.1.4, the PHM has been assumed for this thesis to detect an upcoming failure for an aircraft component at some percentage of the part's actual life, distributed around that percentage as a truncated normal distribution. Both the part life and the detection time are quantified in flight hours, meaning that once a certain number

of flight hours have been flown by the aircraft with the part installed, detection or failure will occur. Once the detection occurs, the part's remaining flight hours until failure are known. This temporal nature of the problem combined with the inherent stochasticity mean that only some of the fleet's aircraft will be detecting failure at any given time, and at some points in time none may be. Thus the fleet can be split into three subsets of aircraft: aircraft which are available to fly and not detecting failure for any installed parts (AC_{Avail}), aircraft which are available to fly but which detect failure for one or more installed parts (AC_{PF}), and aircraft which are unavailable to fly due to needing maintenance for installed parts (AC_U). Only aircraft from the second subset are relevant to the computation of part life wasted, since only these have part lives which can be known to the optimizer. By the same logic, only the second subset is relevant to the computation of maintenance time spacing, since planning maintenance for aircraft which are not predicting failure violates the underlying purpose of maintenance schedule optimization. However, this subset of aircraft does not remain constant through time; as aircraft are repaired and flown they will constantly shift between the three subsets. This in turn means that the optimizer must be run many times over the course of the operational period to continue to make good decisions.

Since the optimizer will only have a limited capacity for making predictions, it must be set up to make decisions based on the current state of the fleet and to assume that this state will change according to predictable rules. Predictable rules include the parts installed on an aircraft degrading as the aircraft flies missions, or the total number of missions flown not being able to exceed the number of missions required. As a result the optimizer will act as if no aircraft may move between subsets without direct action by the optimizer, meaning that an aircraft is able to fly missions until the optimizer decides to schedule maintenance for that aircraft, after which it must

stay in this state[2]. Under this assumption, there is no reason to plan missions and maintenance visits for the entire period of operations for the aircraft; instead, a period restricted between the time the optimizer is run and the furthest-most maintenance visit currently predicted will suffice since all decisions past this point will not involve maintenance scheduling. Finally, since the ideal spacing between maintenance visits is known to be μ_F, this means the time period over which the optimizer will make decisions should end a period $\mu_F * AC_{PF}$ into the future. As mentioned before, the optimizer should be updated whenever new aircraft enter the set AC_{PF}, and the optimization period should be updated accordingly.

Now that the aircraft subsets and time period over which optimization will occur are better understood, the part life wasted can be defined. It was stated before that there are multiple parts installed on each aircraft, and that any number of these parts may simultaneously be detecting failures. Since the need to maintain the aircraft depends solely on the part with the fewest flight hours remaining, the aircraft's flight hours remaining until maintenance must be carried out is the same as for this limiting part. Thus the aircraft in the set of aircraft predicting failure have a value L_j which defines the upper limit of flight hours that may be flown on that aircraft before maintenance must occur. The actual flight hours that will be flown on the aircraft are defined by the number of missions flown over that period and the duration of the mission. Thus the aircraft will ideally fly as many missions as possible before maintaining, with an upper bound dictated by the flight hours remaining to the aircraft. This can be defined mathematically through an objective function and a constraint, where the objective is to minimize the difference between the part life remaining and the flight hours flown on the aircraft before maintenance, and the constraint is not to let this value become negative.

[2]In actuality, the aircraft will not stay in a state of being maintained forever, but the optimizer cannot predict how long maintenance will take because this is dependent on the state of maintenance resources and spare inventory.

For a single aircraft, the objective function can be constructed as minimize $L_j - \sum_{k=1}^{D} m_{j,k}d$ where $m_{j,k}$ is the number of missions flown by aircraft j of AC_{PF} in the subset of aircraft predicting failure and D is the number of days over which the optimization is run. Since the optimization period was previously defined in hours, the number of days to operate can be obtained by dividing the number of hours by 24 and rounding up. Thus for the set of aircraft predicting failure, the total amount of part life wasted over the optimization period can be defined by summing over all the aircraft. The problematic aspect of minimizing a function that can technically return negative values which violate the intention of the problem (if the simulation were to fly more missions than the part can sustain) is counteracted by defining a set of constraints which enforce the nonnegativity of this function. These constraints are defined for each aircraft j of AC_{PF} as $L_j - \sum_{k=1}^{D} m_{j,k}d \geq 0$. The mathematical formulation is shown in Equation 8.

$$
\begin{aligned}
\text{given} \quad & AC_{PF} && \text{(number of aircraft predicting failure)} \\
& j && \text{(index of aircraft from 1 to } AC_{PF}\text{)} \\
& D = \left\lceil \frac{\mu_F * AC_{PF}}{24} \right\rceil && \text{(optimization period in days)} \\
& k && \text{(index of days from 1 to D)} \\
& m_{j,k} && \text{(missions flown by aircraft j on day k)} \\
& L_j && \text{(flight hours remaining on aircraft j)} \\
& d && \text{(mission duration)} \\
\text{minimize} \quad & \sum_{j=1}^{AC_{PF}} \left[L_j - \sum_{k=1}^{D} m_{j,k}d \right] && \\
\text{subject to} \quad & L_j - \sum_{k=1}^{D} m_{j,k}d \geq 0 && \forall\, j
\end{aligned}
\tag{8}
$$

Here it is beneficial to have a brief discussion about mission duration. Recall that

the mission duration is a stochastic value based on a triangular distribution. Since this value will not be realized until the mission is actually flown, the optimizer cannot know the actual duration of missions that will be flown in the future. Instead, an assumed value must be used. Furthermore, since a guiding principle behind ALIS is to prevent any unplanned maintenance events, it seems reasonable that it is more important to replace parts before they break than to try to get a small amount of additional part life from them. Therefore the optimizer will assume the mission duration to equal the maximum possible mission duration. As the optimizer is re-run for changing subsets of aircraft, the true part life remaining will update and should help the optimizer to approach a true prediction value.

The two objective functions and set of constraints in Equation 7 through 8 form the core of the goal setting for the optimization problem, but other practical concerns must be considered as well. For instance, with the optimizer controlling the missions that are flown, it must be aware of any previously existing requirements for missions such as the required operational tempo for the fleet. This tempo provides a required number of missions to be flown each day, though if no aircraft are available some missions may go unflown and lead to backlog. Because the operational tempo cannot be set as a hard requirement (constraint) for the optimization problem, another pair of objective function and constraints can be constructed to lead the problem towards favorable solutions. If the operational tempo is defined as OT_k, the total number of missions that must be flown by the fleet on day k, the objective function and constraints pair can be defined as shown in Equation 9.

$$
\begin{aligned}
\text{given} \quad & AC_{PF} && \text{(number of aircraft predicting failure)} \\
& j' && \text{(index of aircraft from 1 to } AC_{Avail}\text{)} \\
& D = \left\lceil \frac{\mu_F * AC_{PF}}{24} \right\rceil && \text{(optimization period in days)} \\
& k && \text{(index of days from 1 to D)} \\
& m_{j',k} && \text{(missions flown by aircraft j' on day k)} \\
& OT_k && \text{(operational tempo of the fleet on day k)} \\
\text{maximize} \quad & \sum_{j'=1}^{AC_{Avail}} \sum_{k=1}^{D} m_{j',k} \\
\text{subject to} \quad & OT_k - \sum_{j'=1}^{AC_{Avail}} m_{j',k} \geq 0 && \forall\, k
\end{aligned}
\tag{9}
$$

Two more practical concerns must be accounted for when considering the optimization in the context of an evolving simulation. The first is the fact that, when the optimization is initialized, some of the available aircraft may have already been selected to fly a mission for the day and may therefore be unavailable to fly more due to the daily mission limit for each aircraft. This leads to Equation 10. The second is the fact that, at certain times the optimization must be re-run when some of the aircraft are currently flying missions and unable to fly more. If this is the case, an additional constraint as in Equation 11 must be added.

$$
\begin{aligned}
\text{given} \quad & j'' && \text{index of aircraft exceeding mission quotas for the day} \\
& m_{j'',0} && \text{missions flown by aircraft } j'' \text{ on first day} \\
\text{subject to} \quad & m_{j'',0} = 0
\end{aligned}
\tag{10}
$$

$$\begin{aligned} \text{given} \quad & j''' && \text{index of aircraft currently flying missions} \\ & m_{j''',0} && \text{missions flown by aircraft } j''' \text{ on first day} \\ \text{subject to} \quad & m_{j''',0} = 0 \end{aligned} \tag{11}$$

Finally, the fact that the first objective function, Equation 7, was defined using subsequent visits to maintenance must be addressed. The indices for Equations 7 and 8 are the same because they include the same subset of aircraft. However, if one were to assign those indices to aircraft in a random fashion, the first objective function would not match the intended purpose. In order to translate this requirement, consider a specific assignment of indices to aircraft by the order in which those aircraft visit maintenance. Since this order is not defined a priori, it must be determined. To do so, one further assumption must be made. As discussed previously, the repair time for aircraft is uncertain due to limited resources. This fact makes it difficult to determine a time when aircraft can safely begin to be assigned to missions during the optimization period once maintenance has been scheduled to occur. To account for this fact, the assumption will be made within the optimization problem that aircraft cannot fly during the optimization period once maintenance has been scheduled for them. Keep in mind that, due to the frequency with which the optimization will be re-run based on new information, this fact is not expected to lead to aircraft being underutilized. Instead, it makes it much easier to avoid violating the requirements of the maintenance loop as well as making it possible to define a function that enforces the ordering of maintenance times.

To do this, first define a set of decision variables that assign the j aircraft in AC_{PF} to the AC_{PF} subsequent maintenance times. The variable matching aircraft j to maintenance time i, $a_{i,j}$ equals 1 if the aircraft is fixed in the i^{th} maintenance slot and 0 otherwise. Since no two aircraft can be fixed exactly simultaneously, the sum of these decision variables over j for any given maintenance slot i should be

1, and since no aircraft can be assigned to multiple maintenance slots in the same optimization period the same is true of the sum over i for any given j. These are codified in Equation 12.

$$\begin{aligned} \text{given} \quad & AC_{PF} && \text{(number of aircraft predicting failure)} \\ & i && \text{(index of maintenance slots from 1 to } AC_{PF}\text{)} \\ & j && \text{(index of aircraft from 1 to } AC_{PF}\text{)} \\ \text{subject to} \quad & \sum_{j=1}^{AC_{PF}} a_{i,j} && \forall\, i \\ & \sum_{i=1}^{AC_{PF}} a_{i,j} && \forall\, j \end{aligned} \tag{12}$$

These variables are used to assign the aircraft to maintenance visits, but without one further constraint this assignment would have no impact on the optimization problem because its decision variables do not appear in any objective functions. However, there is another condition to enforce which specifies the assumption already listed that an aircraft cannot fly any missions after visiting maintenance. Rewording this, maintenance must be carried out after the aircraft's last mission. This leads to one final function that must be defined. It is easy for a person to look at the set of missions and determine the last day an aircraft flew a mission. However, to define this in mathematical terms is less trivial. Logically speaking, if one looks at all the variables $m_{j,k}$ for a given aircraft j, the $m_{j,k}$ with the highest index k which is nonzero tells us that the last mission flown by aircraft j occurs on day k. To return this value k for the first nonzero mission day requires that all higher and lower values of k be multiplied by zero if no mission was flown. A function which achieves this for a given aircraft j is the indicator function $\xi = \sum_{k=1}^{D} k m_{j,k} 0^{\sum_{k'=k+1}^{D} m_{j,k'}}$. Looking at the second and third multiplication terms, the only day on which the addition term should be nonzero is the day on which $m_{j,k} = 1$ and on which $0^{\sum m_{j,k' \geq k}} = 0^0 = 1$. This means that the

final constraint can now be defined. The constraint specifies that the maintenance event i which aircraft j is assigned to must occur after the final mission flown by this aircraft. The time at which the aircraft is finished with the final mission can be defined as $24(\xi_j - 1) + ST + PT + d$ where ST is the hour of the day on which missions begin to be assigned, and PT is the prep time required to ready an aircraft to fly a mission. The prep time is stochastic, as it requires the subsequent completion of a series of steps which are either distribution-based or which depend on the presence of limited resources. However, for the same reason that the optimization assumed a value for the duration of the mission d, in this case the optimization will ask the simulation its most recent value for PT, which will be computed as a rolling average of the prep time over the previous day's flights. The final version of the constraint is presented in Equation 13.

$$
\begin{aligned}
\text{given} \quad & AC_{PF} && \text{(number of aircraft predicting failure)} \\
& i && \text{(index of maintenance slots from 1 to } AC_{PF}\text{)} \\
& j && \text{(index of aircraft from 1 to } AC_{PF}\text{)} \\
& a_{i,j} && \text{(assignment of maintenance slot i to aircraft j)} \\
& T_i && \text{(time of } i^{th} maintenancevisit\text{)} \\
& D = \left\lceil \frac{\mu_F * AC_{PF}}{24} \right\rceil && \text{(optimization period in days)} \quad (13) \\
& k && \text{(index of days from 1 to D)} \\
& m_{j,k} && \text{(missions flown by aircraft j on day k)} \\
& ST && \text{(time of day that missions start to be flown)} \\
& PT && \text{(rolling average of mission prep time)} \\
& d && \text{(maximum mission duration in hours)}
\end{aligned}
$$

subject to

$$\sum_{j=1}^{AC_{PF}} (a_{i,j}T_i) - 24\left[\sum_{k=1}^{D} km_{j,k}0^{\sum_{k'=k+1}^{D} m_{j,k'}} - 1\right] - ST - PT - d \geq 0$$

$$\forall\, i \in AC_{PF}$$

The optimization problem as currently written is shown in Equation 14. It is multi-objective and non-linear, and allows a great deal of freedom in choosing which aircraft to operate and maintain and when these things occur. The practical aspects of implementing this type of optimization problem will be discussed in Section 3.3.2.

given

AC_{PF} (number of aircraft predicting failure)

AC_{Avail} (number of aircraft available to fly missions)

i (index of maintenance slots from 1 to AC_{PF})

j (index of aircraft from 1 to AC_{PF})

j' (index of aircraft from 1 to AC_{Avail})

j'' (index of aircraft from 1 to number exceeding daily missions)

j''' (index of aircraft from 1 to number currently unavailable)

$D = \left\lceil \frac{\mu_F * AC_{PF}}{24} \right\rceil$ (optimization period in days)

k (index of days from 1 to D)

T_i (Time of i^{th} maintenance slot)

μ_F (mean time between failures for the fleet)

L_j (flight hours remaining to aircraft j)

$m_{j,k}$ (missions flown by aircraft j on day k) (14)

d (maximum mission duration)

$m_{j',k}$ (missions flown by aircraft j' on day k)

OT_k (operational tempo on day k)

$m_{j'',0}$ (missions flown by aircraft j'' on day k)

$m_{j''',0}$ (missions flown by aircraft j''' on day k)

$a_{i,j}$ (assignment of maintenance slot i to aircraft j)

ST (time of day missions start being flown)

PT (mission preparation time)

minimize

$\sum_{i=0}^{n} |T_{i+1} - T_i - \mu_F|$

$\sum_{j=1}^{AC_{PF}} \left[L_j - \sum_{k=1}^{D} m_{j,k} d \right]$

maximize

$\sum_{j'=1}^{AC_{Avail}} \sum_{k=1}^{D} m_{j',k}$

subject to

$L_j - \sum_{k=1}^{D} m_{j,k} d \geq 0 \, \forall \, j \in AC_{PF}$

$OT_k - \sum_{j=1}^{AC_{Avail}} m_{j,k} \geq 0 \, \forall \, k \in D$

$m_{j'',0} = 0 \, \forall \, j''$ exceeding daily missions

$m_{j''',0} = 0 \, \forall \, j'''$ currently unavailable

$\sum_{j=1}^{AC_{PF}} a_{i,j} \, \forall \, i \in AC_{PF}$

$\sum_{i=1}^{AC_{PF}} a_{i,j} \, \forall \, j \in AC_{PF}$

$\sum_{j=1}^{AC_{PF}} (a_{i,j} T_i) - 24 \left[\sum_{k=1}^{D} k m_{j,k} 0^{\sum_{k'=k+1}^{D} m_{j,k'}} - 1 \right] - ST - PT - d \geq 0$

$\forall \, i \in AC_{PF}$

$0 \leq m_{j',k} \leq 1 \, \forall j' \in AC_{Avail}, k \in D$

$0 \leq a_{i,j} \leq 1 \, \forall i, j \in AC_{PF}$

$0 \leq T_i \leq 24D \, \forall i \in AC_{PF}$

$m_{j',k}, a_{i,j}$ integer

3.3.2 Optimization Implementation

Though Equation 14 is complete and makes only one simplifying assumption, it creates some difficulties for implementation within a simulation that represents the operational history of a fleet over the course of a year or more. Optimization solution time scales which are trivial in the real world (on the order of seconds to minutes) create significantly more difficulties for a simulation which calls the optimization many times and which provides answers in seconds. The problem as stated, because it is multi-objective and nonlinear in several of the constraints, and therefore cannot be solved through the preferred mixed-integer linear program, requires the use of some form of stochastic optimizer to solve. However, stochastic optimizers take on the order of seconds to minutes to run and do not provide consistent answers on multiple repetitions of the same problem. Also, these methods have a more difficult time with the strict enforcement of constraints. For this reason it would be better to use a

MILP which would start with a fesible problem and then optimize, always return the same answer, and would run on the order of milliseconds.

Formulating the problem in this manner requires one further assumption. Since the main source of nonlinearity is based on the requirement that missions not be flown after the aircraft is maintained due to the need to determine the final mission day and assign aircraft to maintenance times, these requirements can be removed by assuming that aircraft are maintained in order of those with the fewest flight hours remaining to those with the most flight hours remaining. This removes some degrees of freedom from the problem, but since the optimization must assume that all missions take the maximum mission duration to run, aircraft are largely interchangeable when it comes to the part life wasted objective function. It is therefore not expected that this assumption will reduce the freedom of the optimization method to fly all missions with the least part life wasted. By making this assumption, however, all constraints can be linearized. However, this requires a slight redefinition of the design variables.

If aircraft are ordered by the number of flight hours remaining to them, the set of aircraft AC_i from 1 to AC_{PF} is assigned to the T_i maintenance times by index number. To enforce the requirement that maintenance occur after the aircraft has flown its final mission of the optimization period, the day on which each aircraft visits maintenance is computed from the assumption that these maintenance times will be occur with close to the ideal spacing value. Thus the maintenance day for aircraft i is computed as $MD_i = \mu_F i/24$ when rounded up to the next whole number. The value MD_i is then used to determine the bounds for when the maintenance visits can occur and when missions can be flown.

In removing the assignment of aircraft to maintenance times, the constraints associated with those decision variables disappear. However, one final pair of constraints must be added to address the absolute value used in the first objective function using a method introduced in Linear Programming: A Concise Introduction [40]. The

absolute value function is nonlinear, but when used in a minimization problem can be replaced with a variable ζ and two additional constraints. ζ replaces the entire absolute value expression as follows: $\zeta_i = T_i - T_{i-1} - \mu_F$. To enforce the absolute value criterion, two constraints are added as $\zeta_i - (T_i - T_{i-1} - \mu_F) \geq 0 \, \forall i$ and $\zeta_i + (T_i - T_{i-1} - \mu_F) \geq 0 \, \forall i$.

Due to the multiple objectives, an OEC in the form of a weighted sum must be used to form a single objective function as discussed in Chapter 2 and as suggested in the literature[33, 17]. To ensure that none of the objectives is unfairly weighted, the individual objectives must first be normalized. Normalization requires dividing each objective by its maximum possible value so that it is scaled from zero to one. This is simpler in some cases than in others. For the objective function in Equation 9, the maximum possible value is the sum of the operational tempo over the time period of optimization, $\sum_{j=1}^{D} OT_j$. For the objective function in Equation 8, the maximum possible value is the sum of the part life remaining at the beginning of the optimization period over all aircraft predicting failure, $\sum_{i=1}^{AC_{PF}} L_i$. However, the maintenance interval objective seeks to make the interval close to a target value μ_F. Depending on the scenario, spacing of less than μ_F or greater than μ_F could be larger. This question is explored in more detail in Figures 18 through 20 and the supporting equations.

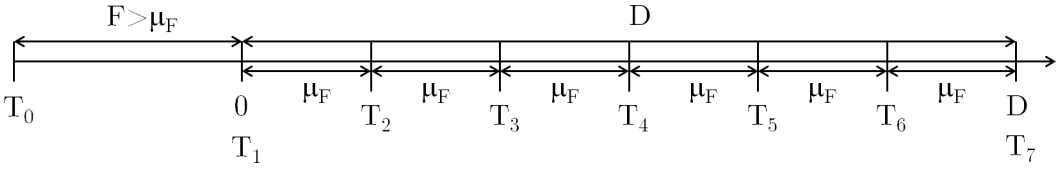


Figure 18: Notional timeline of evenly spaced maintenance visits T_0 through T_7

Figure 18 shows one possible way the maintenance visits can be spaced. In this case, they are at the optimum value, where every interval from $T_2 - T_1$ through $T_7 - T_6$ is equal to μ_F ($T_1 - T_0$ cannot possibly be any closer to μ_F in this case). F is the distance between the last scheduled maintenance event, T_0, and the beginning of the optimization period at zero. However, the value of interest is the maximum value.

Figures 19 and 20 show the two possibilities for the maximum maintenance spacing.

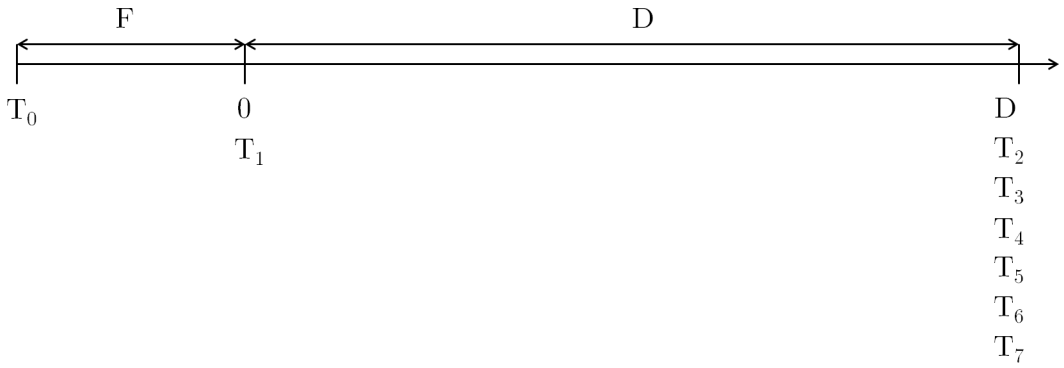


Figure 19: Notional timeline of unevently spaced maintenance visits T_0 through T_7

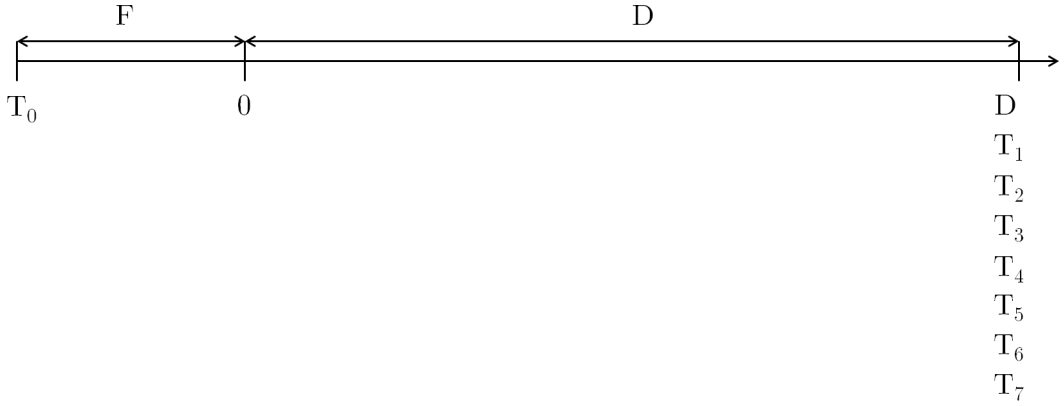


Figure 20: Notional timeline of unevently spaced maintenance visits T_0 through T_7

In Figure 19, one good possibility for the maximum of Equation7 is shown. This would theoretically maximize the objective because, discounting T_0, the maximum amount any two subsequent maintenance events T_{i+1} and T_i can be separated by within the optimization period is D. At this separation value, the contribution to the objective function from those two maintenance visits is $D - \mu_F$. Since the rest of the maintenance visits by definition must occur at either time 0 or time D, as is shown in Figure 19 the contribution from the remaining maintenance events is $(AC_{PF} - 2)\mu_F$, or in this case $5\mu_F$. Thus for the scenario shown in Figure 19, the maximum possible value of the objective function is $D - \mu F + (AC_{PF} - 2)\mu_F$, or $D + (AC_{PF} - 3)\mu_F$, or $D + 4\mu_F$. While still discounting the T_0 maintenance event, the scenario shown in

Figure 20 yields a maximum value of $(AC_{PF} - 1)\mu_F$, or in this case $6\mu_F$. Assuming that $D > 2\mu_F$, the spacing shown in Figure 19 maximizes the objective function.

However, when one accounts for T_0, the other scenario yields a more maximum value. Factoring in T_0, Figure 19's maximum value objective function is $F + D + (AC_{PF} - 3)\mu_F$. However, Figure 20's maximum value objective function is $F + D + (AC_{PF} - 1)\mu_F$, which is categorically greater. Thus the maximum possible value for Equation 7 is $F + D + (AC_{PF} - 1)\mu_F$. Taken with the maximum possible values for Equations 8 and 9, the OEC can be developed as shown in Equation 15. This optimization problem fits the requirements of a MILP.

given

AC_{PF} (number of aircraft predicting failure)

AC_{Avail} (number of aircraft available to fly missions)

i (index of aircraft from 1 to AC_{PF})

i' (index of aircraft from 1 to AC_{Avail})

i'' (index of aircraft from 1 to number exceeding daily missions)

i''' (index of aircraft from 1 to number currently unavailable)

$D = \left\lceil \frac{\mu_F * AC_{PF}}{24} \right\rceil$ (optimization period in days)

$MD_i = \frac{\mu_F i}{24}$

ζ_i (dummy variable for aircraft i)

j (index of days from 1 to D)

T_i (Time of i^{th} maintenance slot)

μ_F (mean time between failures for the fleet)

L_i (flight hours remaining to aircraft i)

$m_{i,j}$ (missions flown by aircraft i on day j)

d (maximum mission duration)

$m_{i',j}$ (missions flown by aircraft i' on day j)

OT_j (operational tempo on day j)

$m_{i'',0}$ (missions flown by aircraft i'' on day j)

$m_{i''',0}$ (missions flown by aircraft i''' on day j)

ST (time of day missions start being flown)

PT (mission preparation time)

W_1 (OEC weighting value on objective 1)

W_2 (OEC weighting value on objective 2)

W_3 (OEC weighting value on objective 3)

F (time between T_0 and beginning of optimization period)

(15)

minimize

$$W_1 \frac{\sum_{i=1}^{AC_{PF}} \zeta_i}{F+D+(AC_{PF}-1)\mu_F} + W_2 \frac{\sum_{i=1}^{AC_{PF}} \left[L_i - \sum_{j=1}^{MD_i} m_{i,j} d\right]}{\sum_{i=1}^{AC_{PF}} L_i} + W_3 \frac{\sum_{i'=1}^{AC_{Avail}} \sum_{j=1}^{\min(MD_i, D)} - m_{i',j}}{\sum_{j=1}^{D} OT_j}$$

subject to

$$L_i - \sum_{j=1}^{MD_i} m_{i,j} d \geq 0 \, \forall \, i \in AC_{PF}$$

$$OT_j - \sum_{i=1}^{AC_{Avail}} m_{i,j} \geq 0 \, \forall \, j \in MD_i$$

$m_{i'',0} = 0 \, \forall \, i''$ exceeding daily missions

$m_{i''',0} = 0 \, \forall \, i'''$ currently unavailable

$$\zeta_i - (T_i - T_{i-1} - \mu F) \geq 0 \, \forall i \in AC_{PF}$$

$$\zeta_i + (T_i - T_{i-1} - \mu F) \geq 0 \, \forall i \in AC_{PF}$$

$$0 \leq m_{i',j} \leq 1 \, \forall i' \in AC_{Avail}, j \in min(MD_i, D)$$

$$24 * (MD_i - 1) + ST + PT + d \leq T_i \leq 24 MD_i + ST + PT + d \, \forall i \in AC_{PF}$$

$m_{i',j}$ integer

Because the multi-objective aspect of the problem was handled by performing a simple summation, the weightings of the three objective functions within this summation should be examined for their impact on the result of the optimization. Depending on the values of these weightings, the problem may place stronger preferences on different aspects of the problem and may push the overall behavior of the fleet towards an undesirable situation. For instance, if too strong a preference is placed on minimizing the part life wasted, the evenness of maintenance visits may be completely sacrified in an attempt to satisfy a different and conflicting aspect of the problem. This will be done in Chapter 5. Based on the findings of this study, it may be necessary to use a more sophisticated multi-objective optimization method than a simple weighted sum. If this should be the case, Ehrgott[34, 35] and Mavrotas[82] have proposed alternate methods for performing multi-objective optimization for combinatorial problems.

With this framework in mind, Chapter 4 will discuss the creation of a model to perform the different experiments.

CHAPTER IV

SUSTAIN-ME VERIFICATION

As explained in Chapter 3, the maintenance optimization method will be tested through a virtual experimentation platform. Chapter 4 describes the development of this platform through the buildup of several different elements. Chapter 1 described the major sustainment elements as operations, maintenance, maintenance paradigm, and supply chain. Due to the nature of the DES methodology used for creating Sustain-ME, the modules of the code do not map exactly to these elements. Figure 21 shows the relationship between the four overarching sustainment pieces; in it, there is sometimes overlap of what is modeled in each of the code modules along the sides of the image. This is due to the fact that modules focus on the behavior of *entities* within the code, whereas the phase breakdown presented in Chapter 1 focused on the different *functions* of sustainment. That different entities share these functions is not terribly surprising, but it does make the description of Sustain-ME's development slightly less straightforward.

Figure 22 shows how the code portions presented in Figure 21 were developed modularly by changing the modeled sustainment behavior small amounts at a time. The modules that were added build on those that already existed; for instance, the sortie generation was the first logical element modeled because it does not require inputs from other segments of the code. The development of the sortie generation code is described and verified in Section 4.3. Next the sortie assignment logical was added because it requires the sortie generation logic plus an aircraft available to fly missions. In this case, a very basic version of the fleet was created as a placeholder that could fly missions and do nothing else. The development and verification of this

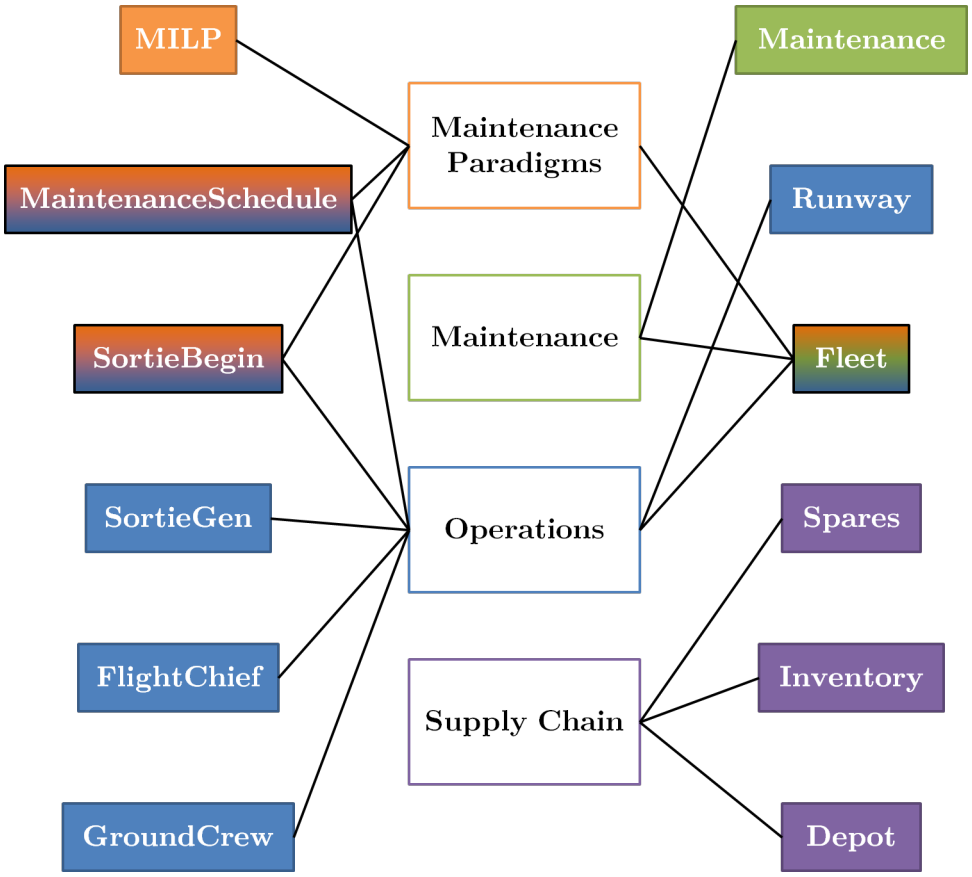


Figure 21: Sustain-ME to sustainment translation

code is described in Section 4.4.

The next logic to be modeled was the rest of the fleet's behavior, which draws from both the operational and maintenance portions of sustainment. In this case, the behavior of the aircraft in the fleet was modeled, but the supply chain elements (spare inventory) were defaulted to the assumption that parts are always available after a delay. The development and verification of this portion is described in Section 4.5. Next the supply chain behavior was added, as described in Section 4.6. At this point a sustainment process under a reactive maintenance paradigm had been modeled since that was the default maintenance behavior used to develop Sustain-ME initially. Once the behavior of sustainment under reactive maintenance had been verified and calibrated, the maintenance paradigm for CBM was modeled using the assumption that a PHM is installed. In Figure 22, this model version appears to be the same as the one where the supply chain was integrated; however, the fleet's logic and behavior was updated to account for the differences due to a CBM paradigm. This is denoted with a star. The logic implemented for this version of the code is documented in Section 4.7. Finally, the behavior associated with CBM-MiMOSA was implemented, and this is documented in Section 4.8.

This method of coding a piece, testing it, and coding some more is known as test-based coding. By utilizing this method, the results of the code at each stage are verified against expected behavior and form a solid foundation upon which future code can be built. By the time Sustain-ME is completed, its behavior should be in line with the intended behavior as laid out in Chapters 2 and 3. This process will largely be based on the output of visualizations to confirm that Sustain-ME's outputs are as expected, though in some instances logical checks against known sequences of events will also be employed. The description of the development of Sustain-ME begins in Section 4.1.

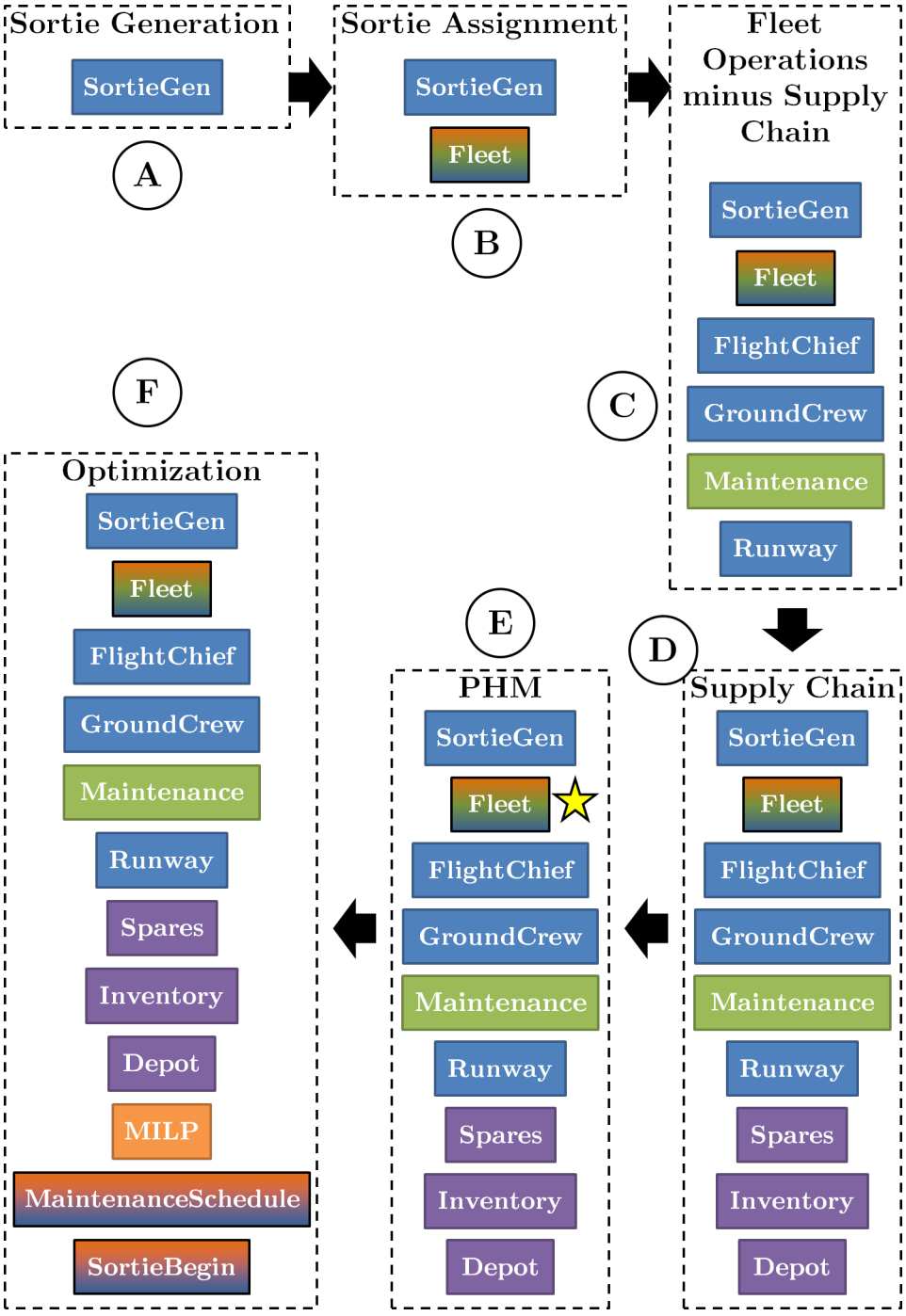


Figure 22: Sustain-ME to sustainment translation

4.1 Inputs

The inputs to Sustain-ME are values for which estimates were not found in the literature, or values whose impact on the model could be interesting enough to warrant study. The values are the operational tempo for sustainment (the number of required missions to fly over the course of the simulation time), the surge profile(a multiplier over time signifying ramp up or down in fleet activities), the reliability of the aircraft (computed in Section 3.1.4), the removal and installation time for LRU's (loosely based off of Faas[39]), PHM detection lead time distribution parameters as a percent of part life, the number of aircraft in the fleet, and the initial inventory investment (the number of spare parts available at the beginning of the simulation). The values used in different aspects of development of Sustain-ME are shown in Table 5.

Table 5: Model inputs for each section

Input	Distribution/Value	Sections Relevant
Operational Tempo	10 Missions/Day	All except 4.3 and
Surge Profile	Flat	All except 4.3
Reliability (Aircraft)	6 hours between repairs	4.3 and 4.5
Reliability (Parts 1&2)	12.5 hours between repairs	4.6
Reliability (Parts 3&4)	300 hours between repairs	
Reliability (Parts 1-4)	25 hours between repairs	4.7 and 4.8
Reliability (Parts 5&6)	300 hours between repairs	
Removal (All parts)	Tri(.75,1.0,1.25)	All
Installation (All parts)	Tri(1.0,1.5,2.0)	All
PHM Detection Time (All parts)	N(0.8,0.05)	4.7, 4.8, and
Fleet size	30 aircraft	All except 4.4
Inventory Investment	Varies with section	

4.2 Assumptions

The assumptions made in creating Sustain-ME were culled from literature and have been described in Chapters 3. These include the distributions associated with various delays in the model, the time of day and day of week during which missions are flown (based on a twelve hour shift, 7 days a week), the time of day that repairs can be

completed (24/7), the daily limit on missions flown (discussed in Section 3.1.4), and the length of time during which unflown required missions are backlogged to be flown again, if possible. These quantities are contained in Table 6.

Table 6: Model assumptions

Assumption	Distribution/Value
Mission Scheduling Duration	Tri(0.5,0.75,1.0) hrs
Preflight Inspection Duration	Tri(0.83,1.0,1.2) hrs
Refuel Duration	Tri(0.33,0.367,0.4167) hrs
Load Weapons Duration	Tri(0.75,1.0,1.25) hrs
Engine Start, Systems Check, & Taxi Duration	Tri(0.1167,0.167,0.2) hrs
Takeoff Duration	Tri(0.033,0.05,0.067) hrs
Sortie Duration Duration	Tri(1.3,1.5,2.0) hrs
Landing Duration	Tri(0.233,0.25,0.267) hrs
Parking & Recovery Duration	Tri(0.0833,0.1167,0.15) hrs
Servicing Duration	Tri(0.75,1.0,1.25) hrs
Document Corrective Actions Duration	Tri(0.0833,0.167,0.25) hrs
Wait for Local Parts Duration	Tri(0.5,0.2,0.25) hrs
Inventory Paperwork Duration	Tri(0.0833,0.167,0.25) hrs
Depot to Base Shipping Duration	Tri(2.4,7.2,12.) hrs
Base to Depot Shipping Duration	U(6.0,12.0) hrs
Depot Repair Duration	Tri(1666.56,2083.2,2499.84) hrs
Flying Hours	7am-7pm
Repair Hours	24/7
Daily Mission Limit	1 per aircraft per day
Backlog Length	7 days

4.3 Sortie Generation: Verification

This section covers the modules denoted by the letter 'A' in Figure 22. Sorties, or missions, are generated according to the rules set out in Sections 4.1 and 4.2. This means the Sortie Generation module uses the weekly operational tempo schedule and surge profile defined in the Inputs module to determine how many missions to fly each day. The backlog rules established in the Assumptions module determine how many days non-completed missions are kept in reserve to be flown if aircraft are still available once completing the current day's missions. The generation of

sorties is performed without reference to the Fleet module, meaning they will continue regardless of aircraft availability.

The verification test for this portion of the code is to output the mission backlog every time it changes along with the time at which it changed. The mission backlog tracks all missions which were required but which have not yet been assigned to an aircraft for the current day and the past n days, where n is the number of days established in the Assumptions module's backlog rules. Examples of this output are shown in Figures 23 through 28 for different backlog rules, op tempos, and surge profiles. The combination of these values is shown in Table 7. In the table the Op Tempo options refer to the number of missions per day and the Surge Profile options refer to either no surge (op tempo remains the same throughout the simulation) or 3-2-1 surge (op tempo multiplied by three for the first seven days, then by two for the next thirty days, then by one for the remainder of the simulation).

Table 7: Sortie generation test cases

Test Case	Backlog	Op Tempo	Surge Profile
Shorter backlog	3 Days	10 Per Day	Don't Surge
Longer backlog	30 Days	10 Per Day	Don't Surge
Baseline	7 Days	10 Per Day	Don't Surge
Baseline w/ surge	7 Days	10 Per Day	3-2-1 Surge
Baseline w/ weekend	7 Days	12 Per Day M-F, 5 Per Day S&S	Don't Surge
Surge with weekend hours	7 Days	12 Per Day M-F, 5 Per Day S&S	3-2-1 Surge

The first test case has a backlog length of three days, an op tempo of ten missions per day and a standard surge profile. The unflown missions over this period of time as seen in Figure 23 show that the backlog for the current day starts at zero, rises to ten missions at 7:00 a.m. on day one of the simulation, and stays at ten missions for the duration of the simulation. The backlog for one day ago exhibits the same behavior but waits until 7:00 a.m. on day two of the simulation to rise to ten missions, and the backlog for two days ago exhibits the same behavior but waits until 7:00 a.m.

on day three of the simulation to rise to ten missions. This confirms a few aspects of the coded model. First, the generation of missions at 7:00 a.m. confirms that the operating hours defined in the assumptions are being adhered to as far as sortie generation is concerned. Second, the correct number of daily missions as defined by the op tempo and surge profile are being generated. Third, unflown missions are being rolled over in the backlog at the beginning of the operational day as specified in the sortie generation module's behavior. This accounts for the delayed rise in the backlog between the current, one day ago, and two days ago entries. Since this version of Sustain-ME does not include any sortie assignment logic or aircraft operational logic, the backlog remains at its maximum value as expected when no missions are being flown.

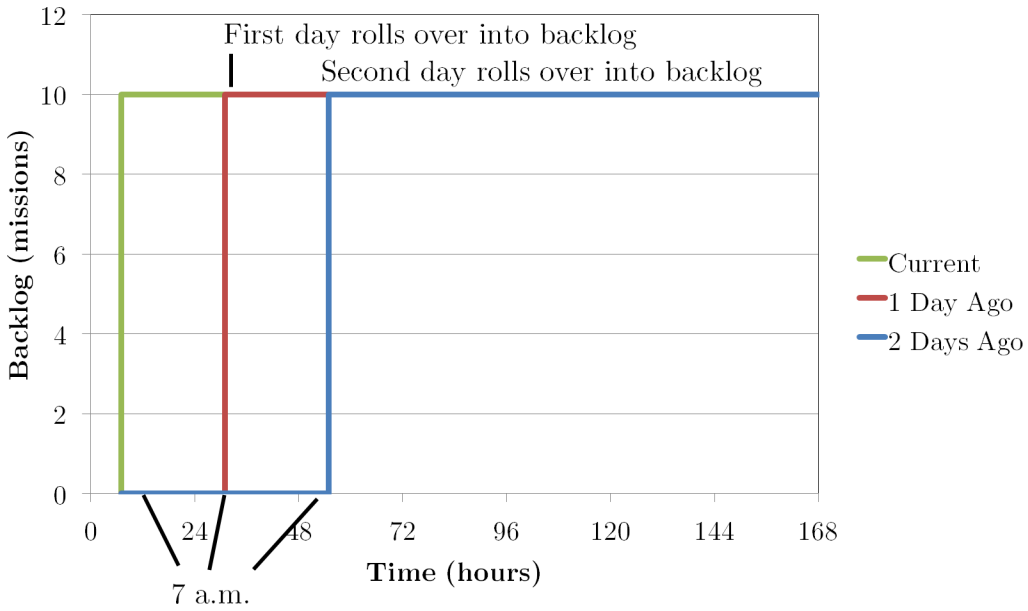


Figure 23: Unflown missions for shorter backlog test case

The second test case has a backlog length of thirty days but is otherwise the same as the first test case. Since the only difference between the two cases is how long unflown missions are stored to be reflown, the behavior for the second case is expected to look the same as for the first case, but with a larger number of previous days to graph. Since the first case showed that each backlog entry has a lag of one

additional day, this behavior should continue with the second case. Figure 24 shows that these expectations are upheld by the code's output. This additionally confirms that Sustain-ME correctly uses the input number of backlog days to increase the length of time that unflown missions are saved to be reflown.

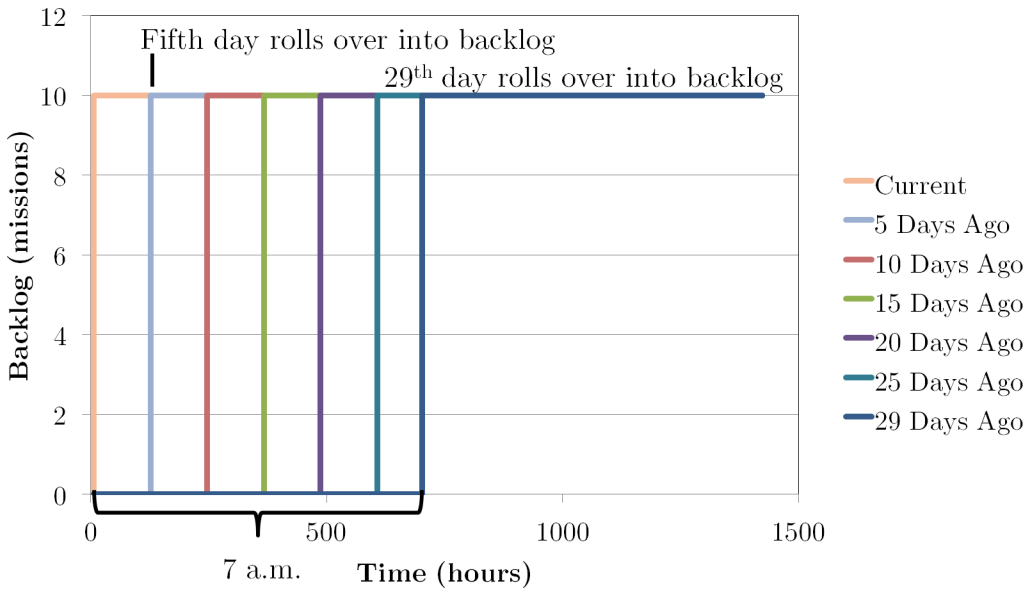


Figure 24: Unflown missions for longer backlog test case

The third test case has a backlog length of seven days and is otherwise the same as the previous two test cases. Figure 25 shows once again that the input number of backlog days is correctly represented in the outputs of Sustain-ME.

The fourth test case has a backlog length of seven days and includes a 3-2-1 surge, but the baseline op tempo (the op tempo when the surge level is one) is still ten missions per day. Since the correct backlog behavior has already been established, Figure 26 shows only the current and six days ago backlog entries for ease of viewing. The figure shows that the current day's unflown missions go from zero to thirty (3×10) at 7:00 a.m. on day one of the simulation, from thirty to twenty (2×10) at 7:00 a.m. on day eight of the simulation, and from twenty to ten at 7:00 a.m. on day thirty-eight of the simulation. The unflown missions from six days ago displays the same behavior, six simulation days after the current day. Taken together, these observations confirm

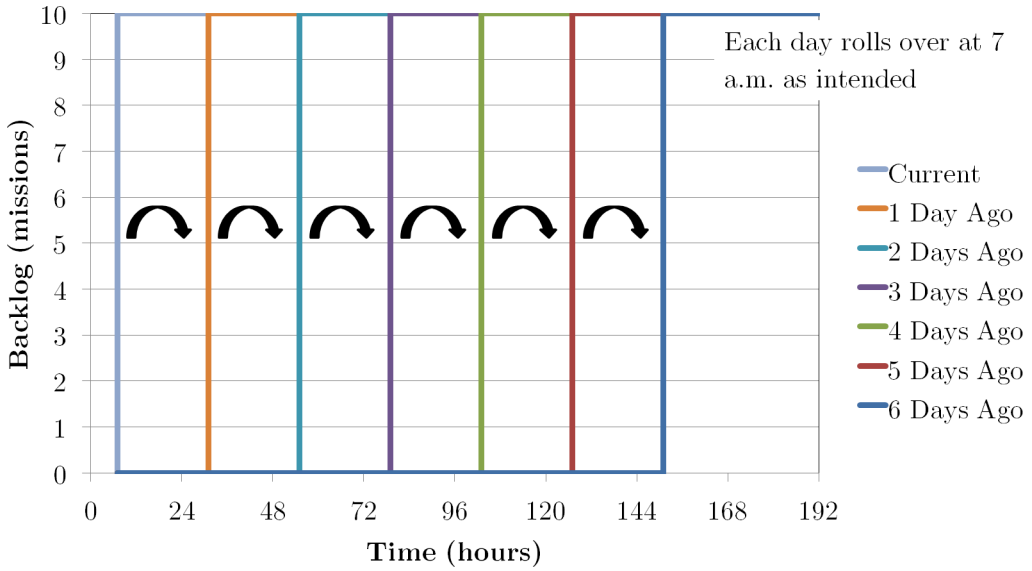


Figure 25: Unflown missions for baseline test case

that Sustain-ME is surging when and how much it was told to, since the times and number of missions for the current day match the behavior described in the 3-2-1 surge model. They also confirm that backlog continues to behave as proscribed, even with the presence of a disruptive element such as surge.

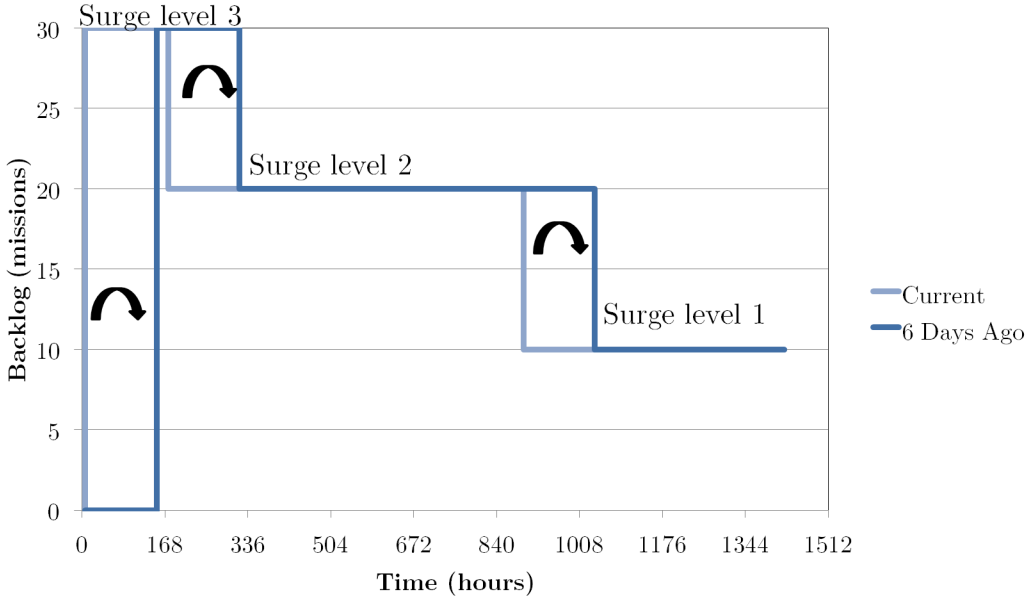


Figure 26: Unflown missions for baseline with surge test case

The fifth test case has a backlog length of seven days and an op tempo of 12

missions per day on Monday through Friday, 5 missions per day Saturday and Sunday with no surge. Figure 27 shows that the current day's backlog goes from zero to five at 7:00 a.m. on day one of the simulation, from five to twelve at 7:00 a.m. on day two of the simulation, from twelve to five at 7:00 a.m. on day seven of the simulation, and then this pattern repeats with five consecutive days spent at 12 missions worth of backlog and two consecutive days spent at 5 missions worth of backlog for the duration of the simulation. Again, this behavior is similar for the 6 days ago backlog entry, but with all times six days after the times listed for the current backlog entry. Taken together, these observations indicate that the varying op tempo is being correctly used by Sustain-ME, since the times listed for the current day line up with Sunday at 7:00 a.m., Monday at 7:00 a.m., and Saturday at 7:00 a.m. respectively. Once again, the backlog behavior remains consistent through the addition of a disruptive element.

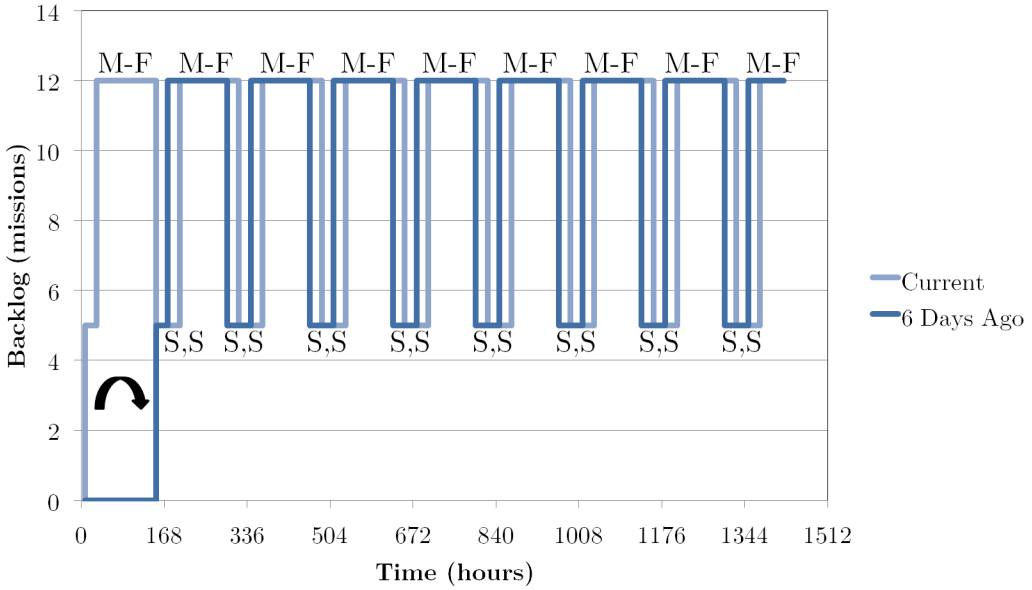


Figure 27: Unflown missions for baseline with weekend hours test case

The sixth test case combines the surge and op tempo behavior described in the fourth and fifth test cases. Since the actions scheduled for op tempo and surge occur at different simulation times, it is helpful to walk through simulated time and show

how these parameters are active at different points during the simulation. This work is shown in Table 8. Because of the more complicated combined schedule between the op tempo and the surge, surge level changes do not always occur at the same time as op tempo mission per day changes. This is reflected in both Table 8 and Figure 28. Because the table values (from a conceptual understanding of the scheduling) match the figure values (from the output of Sustain-ME), the behavior of the model is confirmed for more complicated combinations of the inputs.

Table 8: Scheduled events for surge and weekend test case

Sim Time	Day/Time	Op Tempo Value	Surge Level	Missions
0	Sun 12:00 a.m.	0 missions	3	0
7	Sun 7:00 a.m.	5 missions	3	15
31	Mon 7:00 a.m.	12 missions	3	36
151	Sat 7:00 a.m.	5 missions	3	15
175	Sun 7:00 a.m.	5 missions	2	10
199	Mon 7:00 a.m.	12 missions	2	24
319	Sat 7:00 a.m.	5 missions	2	10
367	Mon 7:00 a.m.	12 missions	2	24
487	Sat 7:00 a.m.	5 missions	2	10
535	Mon 7:00 a.m.	12 missions	2	24
655	Sat 7:00 a.m.	5 missions	2	10
703	Mon 7:00 a.m.	12 missions	2	24
823	Sat 7:00 a.m.	5 missions	2	10
871	Mon 7:00 a.m.	12 missions	2	24
895	Tue 7:00 a.m.	12 missions	1	12
991	Sat 7:00 a.m.	5 missions	1	5
1039	Mon 7:00 a.m.	12 missions	1	12

Having verified that sorties are generated as intended, the assignment of those sorties to simply modeled aircraft can now be verified. This is done in Section 4.4

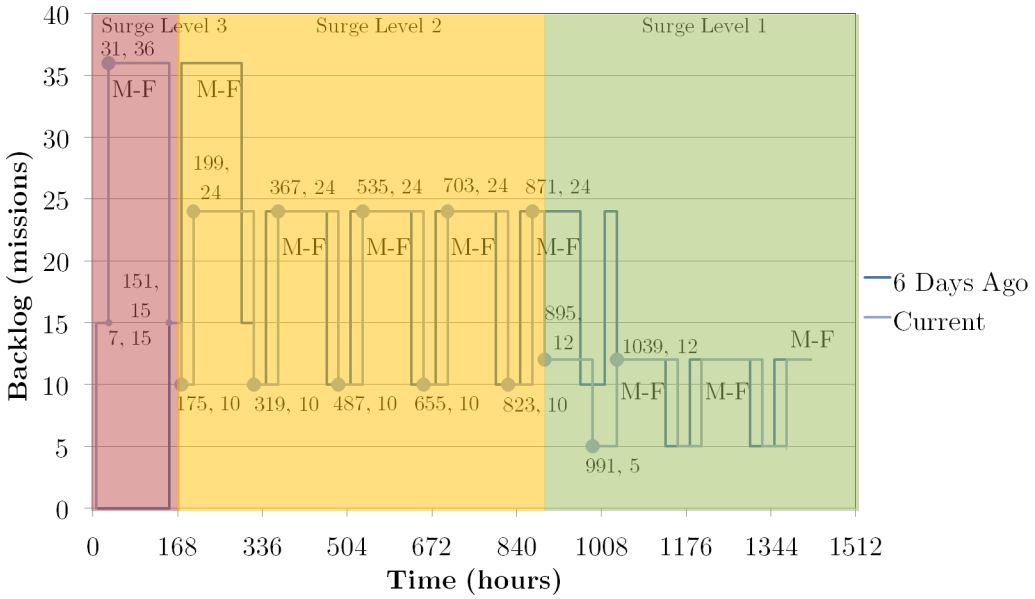


Figure 28: Unflown missions for surge with weekend hours test case

4.4 Sortie Assignment: Verification

This section covers the modules denoted by the letter 'B' in Figure 22. Sorties are assigned according to the rules set out in Section 4.2. This means the Sortie Assignment portion of the Sortie Generation module uses the hours established for flying sorties and the daily mission limit to determine when missions may be flown and which aircraft may fly them. If the current model time is between the hours established for flying missions in the Assumptions module, the Sortie Assignment code will look for aircraft which are available (defined by the Fleet module) and which have not flown more than the assumed daily maximum number of missions. These aircraft will then be specified as unavailable to fly further missions until they have completed the logical loop of the Fleet module. As long as there are missions remaining in the backlog which have not been flown, and the simulation time is between the hours specified for flying sorties, the Sortie Assignment module will attempt to assign aircraft to fly these missions.

The test for this portion of the code is to output the number of available (waiting for missions) aircraft versus simulation time. Aircraft count as available when they

are waiting for a mission, and are not made available again until they have completed the aircraft operations loop. For the Sortie Assignment module test, the default values established for the Sortie Generation test (op tempo of ten missions per day every day, no surge and seven days of backlog) will be used. To determine whether the Sortie Assignment portion is behaving as intended, the results will be displayed for different values of fleet size: ten aircraft, thirty aircraft, and fifty aircraft. Examples of this output are shown in Figures 29 through 31. Each figure shows the first week of simulation time, since the remaining simulation time continues to show the same behavior. Figure 29 shows that each day, all aircraft are assigned to missions at 7:00 a.m. A dummy value of 10 hours was used as the sortie length to provide an offset between when aircraft are made unavailable (no longer waiting for a mission) and when they become available again. The figure reflects this as all aircraft become available at 5:00 p.m. and remain so until 7:00 a.m. the next day. Figure 30 exhibits the same behavior, but since only ten sorties are generated per day, only ten of the aircraft are assigned to missions each day and the remaining twenty aircraft remain available. Figure 31 also follows this behavior, with forty aircraft remaining available every day. It should be noted that the figure does not mean the *same* aircraft are used every day; only that the same number are used every day.

Having confirmed that sorties are assigned as intended to aircraft, the behavior of the fleet flying those sorties can now be verified. This is done in Section 4.5.

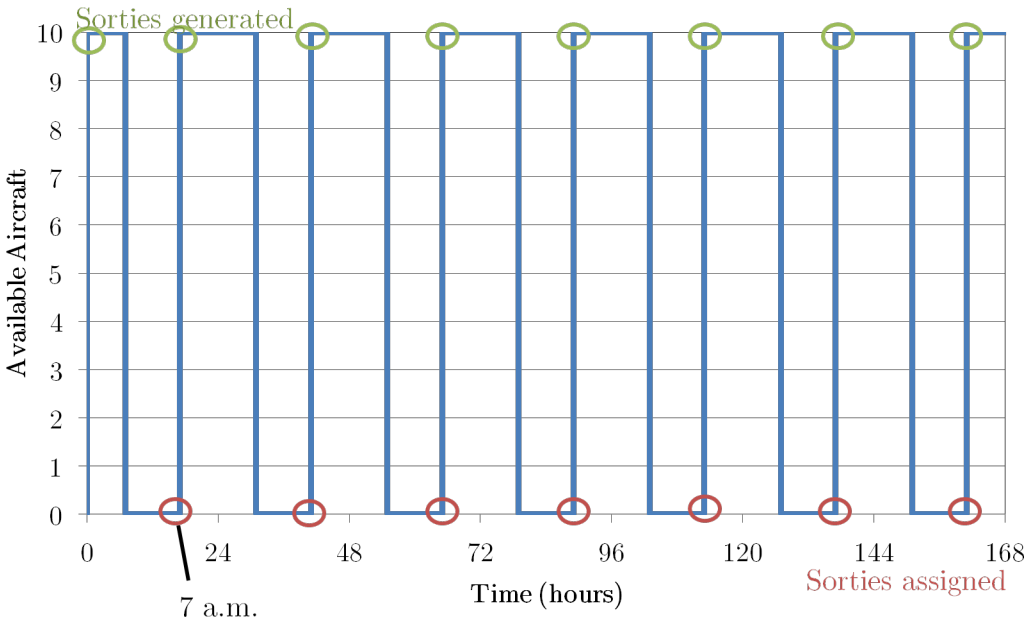


Figure 29: Aircraft awaiting missions – fleet of 10 aircraft

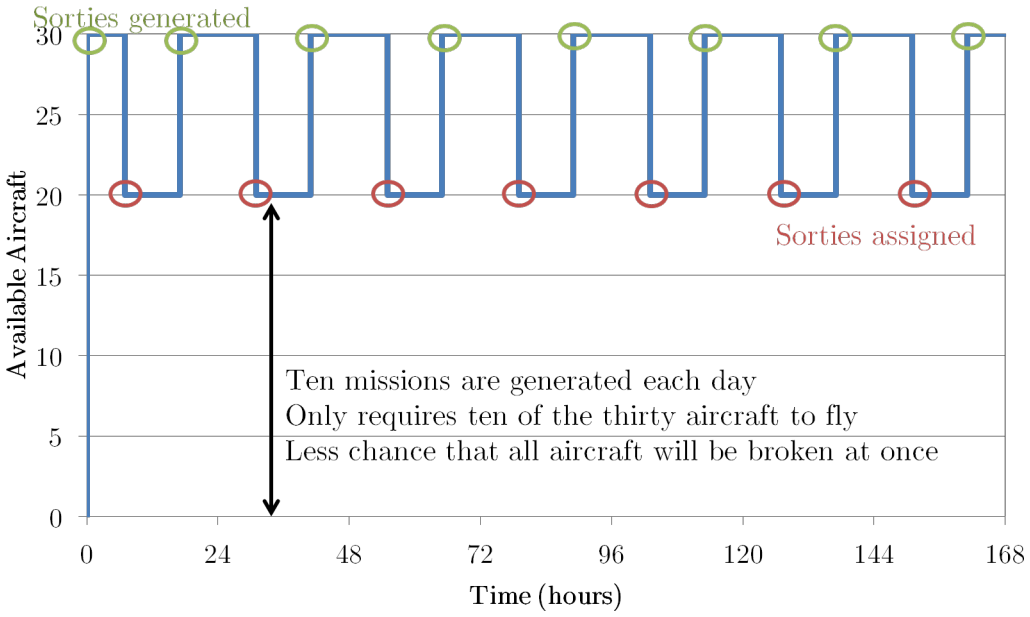


Figure 30: Aircraft awaiting missions – fleet of 30 aircraft

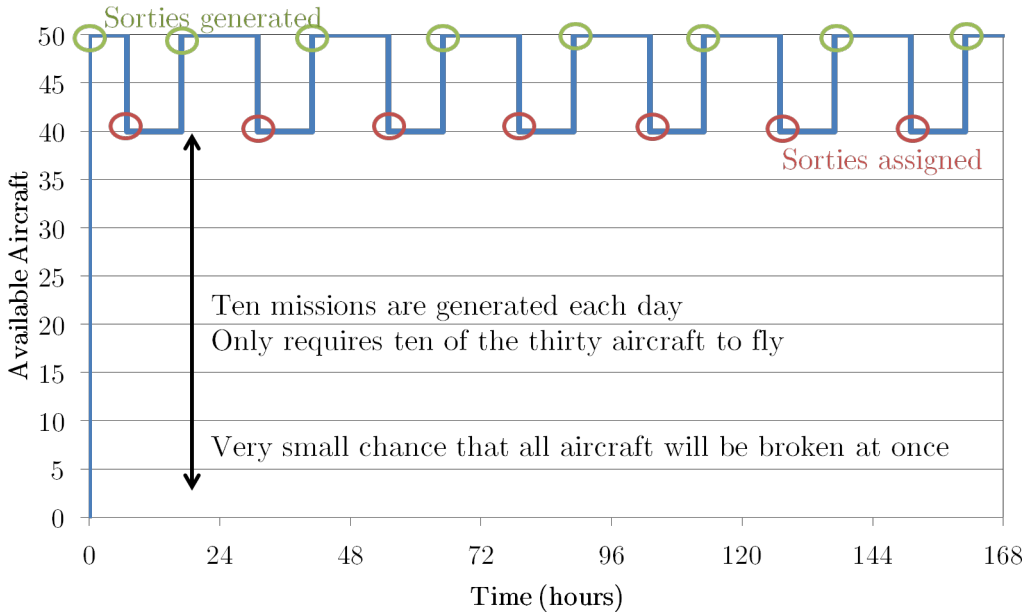


Figure 31: Aircraft awaiting missions – fleet of 50 aircraft

4.5 Fleet Operations Excluding Supply Chain: Verification

This section covers the modules denoted by the letter 'C' in Figure 22. Fleet operations are derived from Figure 32 (which is itself derived from Figures 12 through 14) and the specific distributions that define each step of the process as laid out in Section 4.2. Until the point at which parts are ordered, this process operates independently from the spare parts cycle and interacts only with the Sortie Generation module, so this portion was tested before integrating any supply chain logic. Instead, a minimal wait time for parts was implemented. This represents a best case scenario, where parts are always available when needed. In the real world, this could be achieved by having a far larger than necessary store of spare parts, which is a realistic but undesirable condition. The benefit of running Sustain-ME at this condition when testing is that all negative effects of wait time can be attributed to having insufficient maintenance resouces.

One verification test for this portion of the code is to output the states of the discrete event simulation and the time at which they occur to verify that the logic

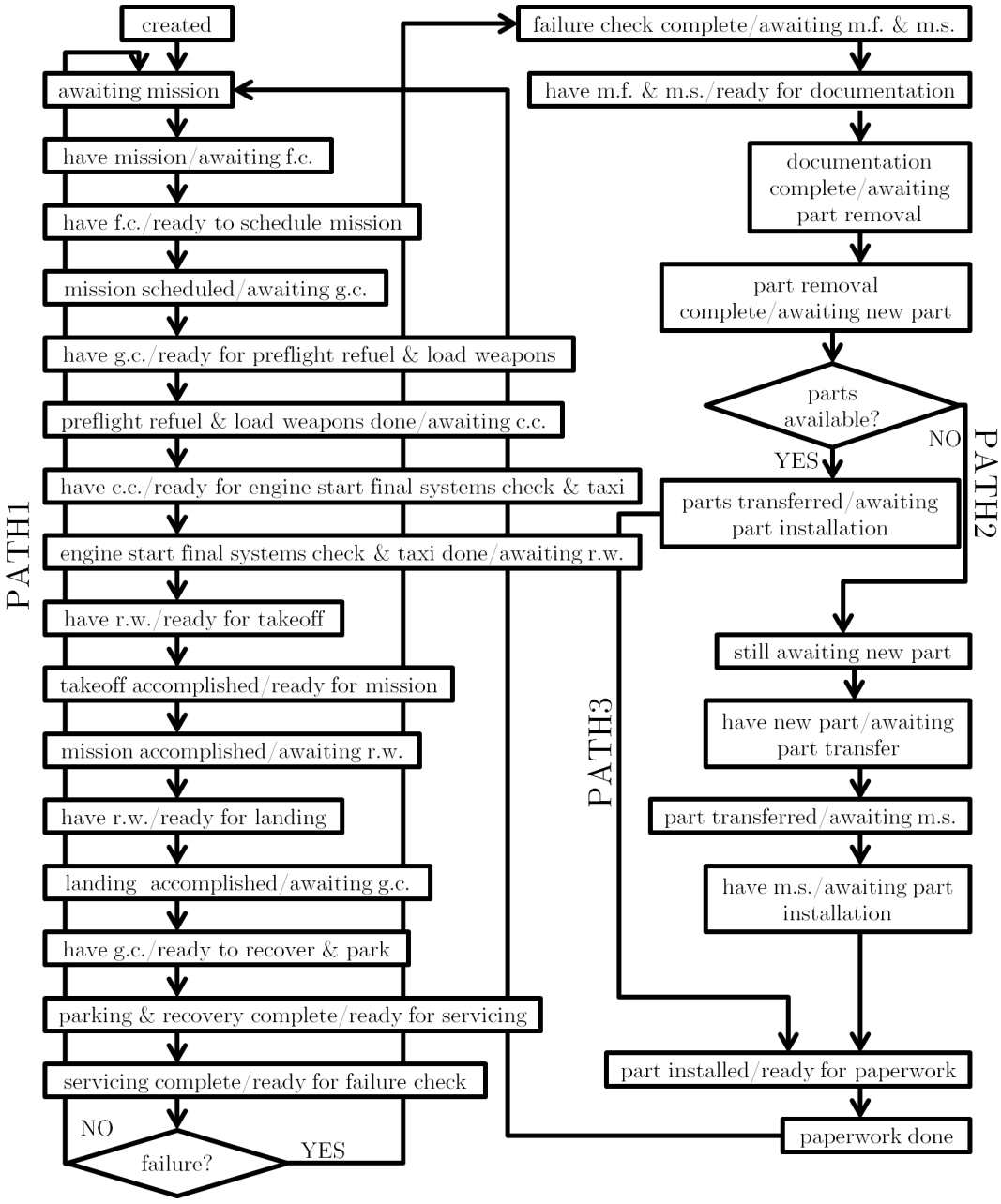


Figure 32: Aircraft order of events flowchart

intended has been implemented. This can be visually examined in two different ways. First, the state of each of the simulation's aircraft can be plotted on the y-axis of a graph vs. the time at which the aircraft is in the given state, known as an event trace diagram. A line on this graph represents the particular sequence of events the aircraft experienced due to the combination of resource levels and draws from random distributions. Figures 33 through 35 show notional paths that aircraft may take; Figure 33 represents what the path looks like if the "NO" path is taken at the first decision point in Figure 32 (the times that the state transitions occur do not reflect the actual delay times from Sustain-ME). Figure 34 represents what the path looks like if the "YES" path is taken at the first decision point and the "NO" path is taken at the second decision point in Figure 32. Figure 35 represents what the path looks like if the "YES" path is taken at both decision points. These graphs are helpful in determining if anything large is wrong in the code, because for the most part the aircraft proceeds through the states one by one and does not move backwards until returning to the state "awaiting mission". If significant delays are introduced for any reason, it will show clearly in the chart and may help to determine remedies for reducing delays. Also, should any aircraft fail to continue scheduling events in the simulation for any reason, this will be evident in the graph as well so long as the x-axis scale is examined and checked against the simulation time.

However, certain errors will not be evident in this plot. Due to the number of states and the small delays between certain states, it will not necessarily be easy to visually match even a single aircraft operations cycle on the chart against the notional plots shown here. This will be especially true when examining the events of an entire simulation. For this reason a secondary check of the aircraft state order will be introduced. This check will compare the event order of individual aircraft against the known possible paths through the simulation. If aircraft deviate from these paths, a value of 1 will be given, and if not a value of 0 will be given. By summing up the check

values over the entire simulation time, a quick assessment can be made of whether the event order matches known and acceptable event paths. This information can also be used to compile data on the percentage of times aircraft followed each of the different paths, which should remain constant given constant part reliability inputs.

The events of the simulation are dependent on a large number of input and assumption variables, notably the large list of distributions listed in Section 4.2. However, these parameters do not need to be varied to verify their correct use in the simulation; instead, over the course of a number of repetitions the duration of each step can be collected and a histogram of all the duration values can be plotted and compared to the input distribution. This represents a secondary test for verifying Sustain-ME. Finally, the operational availability of Sustain-ME will be examined at this state to ensure that it is high, as would be expected when inventory is always available.

4.5.1 Event Activity Verification

The testing begins by examining Sustain-ME's behavior for the minimal resource case to determine that the order of events is correct. Figure 36 shows the evolving state of an aircraft from the fleet over the course of the first thirty simulation days. Each vertical line with endpoints denoted indicates the transition of the aircraft from one state to another, with each state listed on the y-axis corresponding to the state visited. Thus, for the first aircraft operational cycle in Figure 36, the aircraft is created, spends a few hours awaiting a mission (indicated by the visibly horizontal line on the "awaiting mission" line), and then rapidly proceeds through the mission prep, flying, and landing stages of operations. Though this portion of the cycle may appear to be strictly vertical, in reality each step takes a very small amount of time to complete. The aircraft next spends a few hours awaiting the availability of the ground crew so it can complete the rest of the recovery phase. In this case, the aircraft did not have any part failures and therefore did not require maintenance, so it returns to

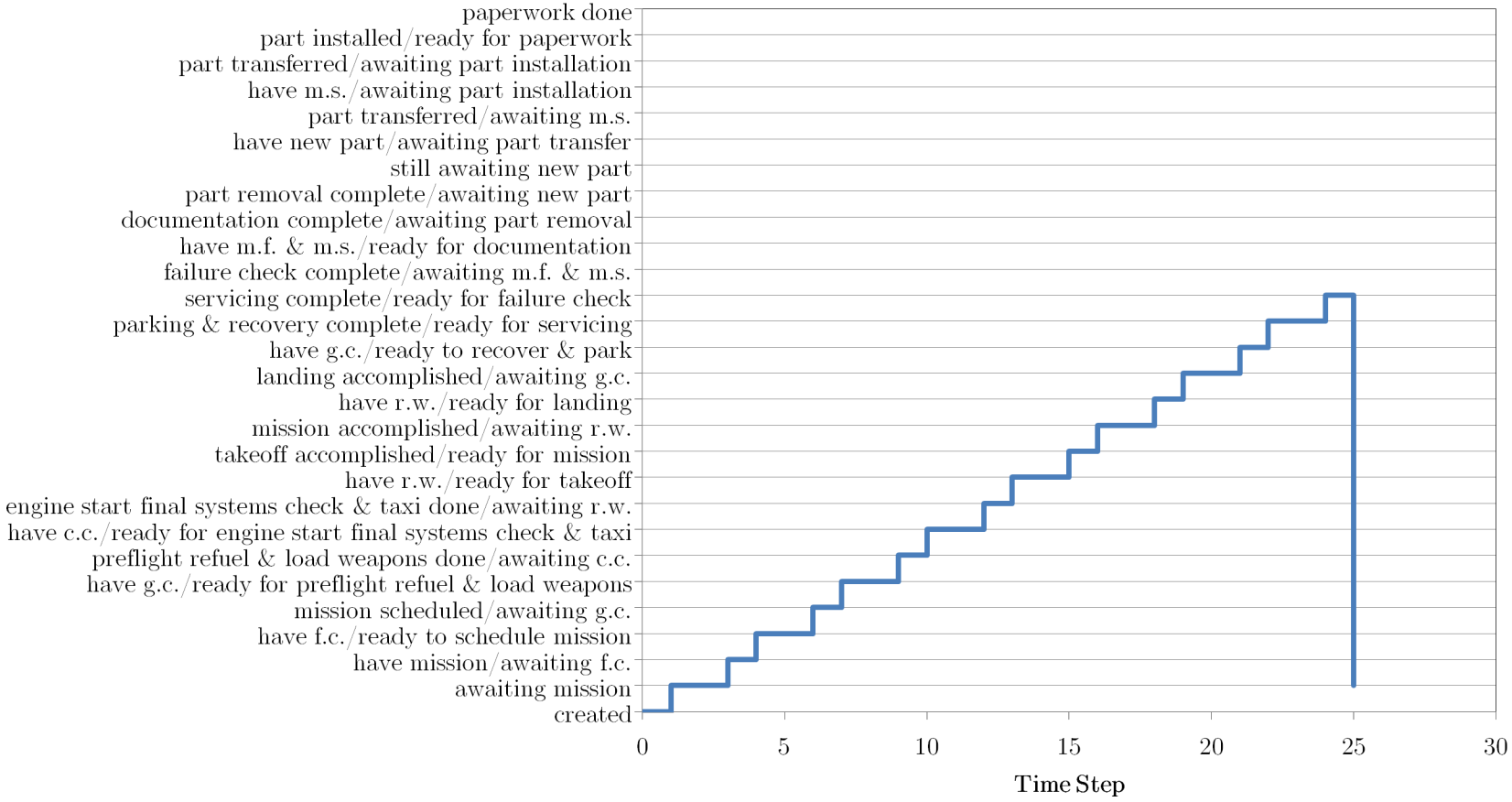


Figure 33: Path 1 event trace diagram

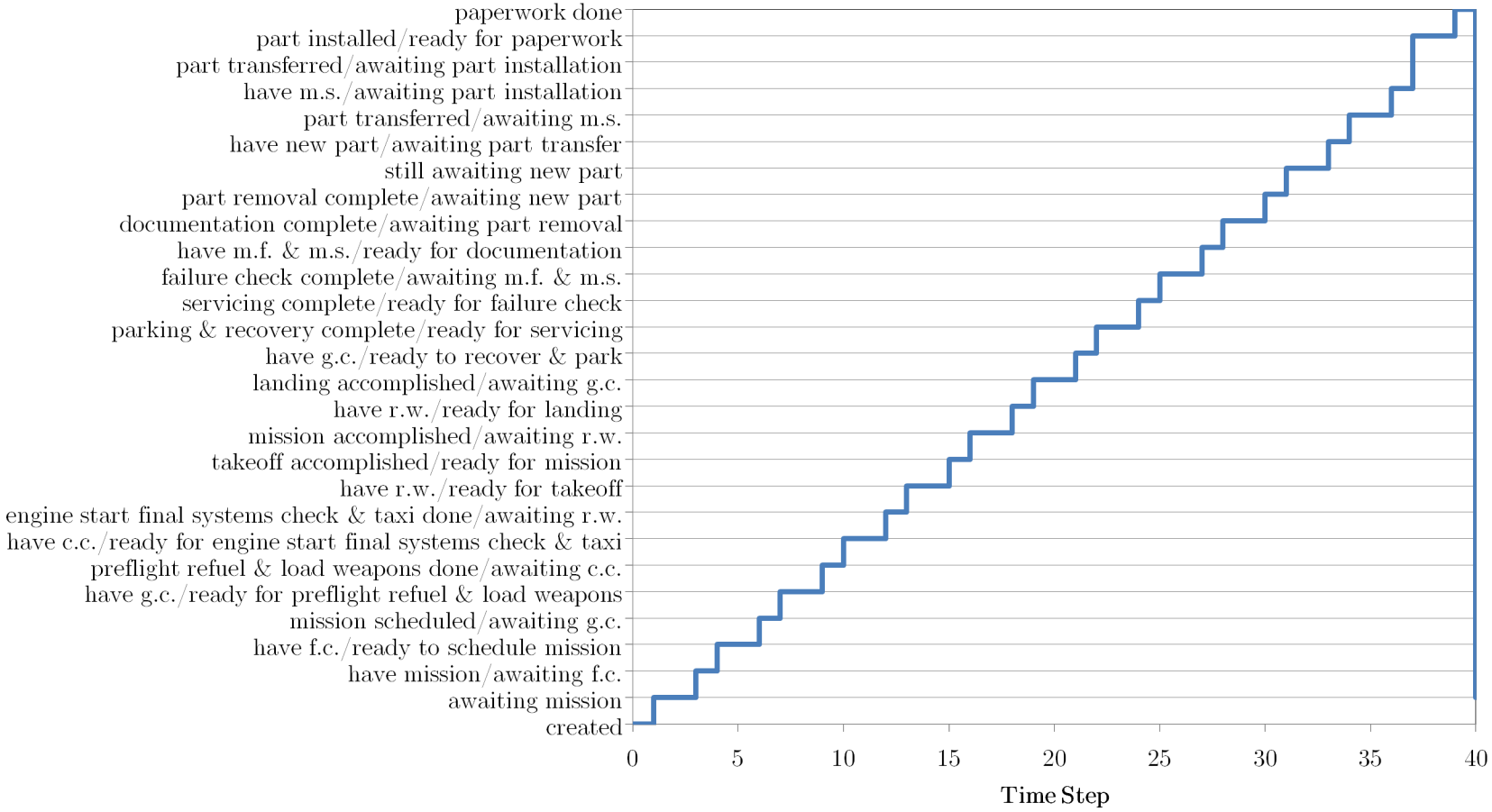


Figure 34: Path 2 event trace diagram

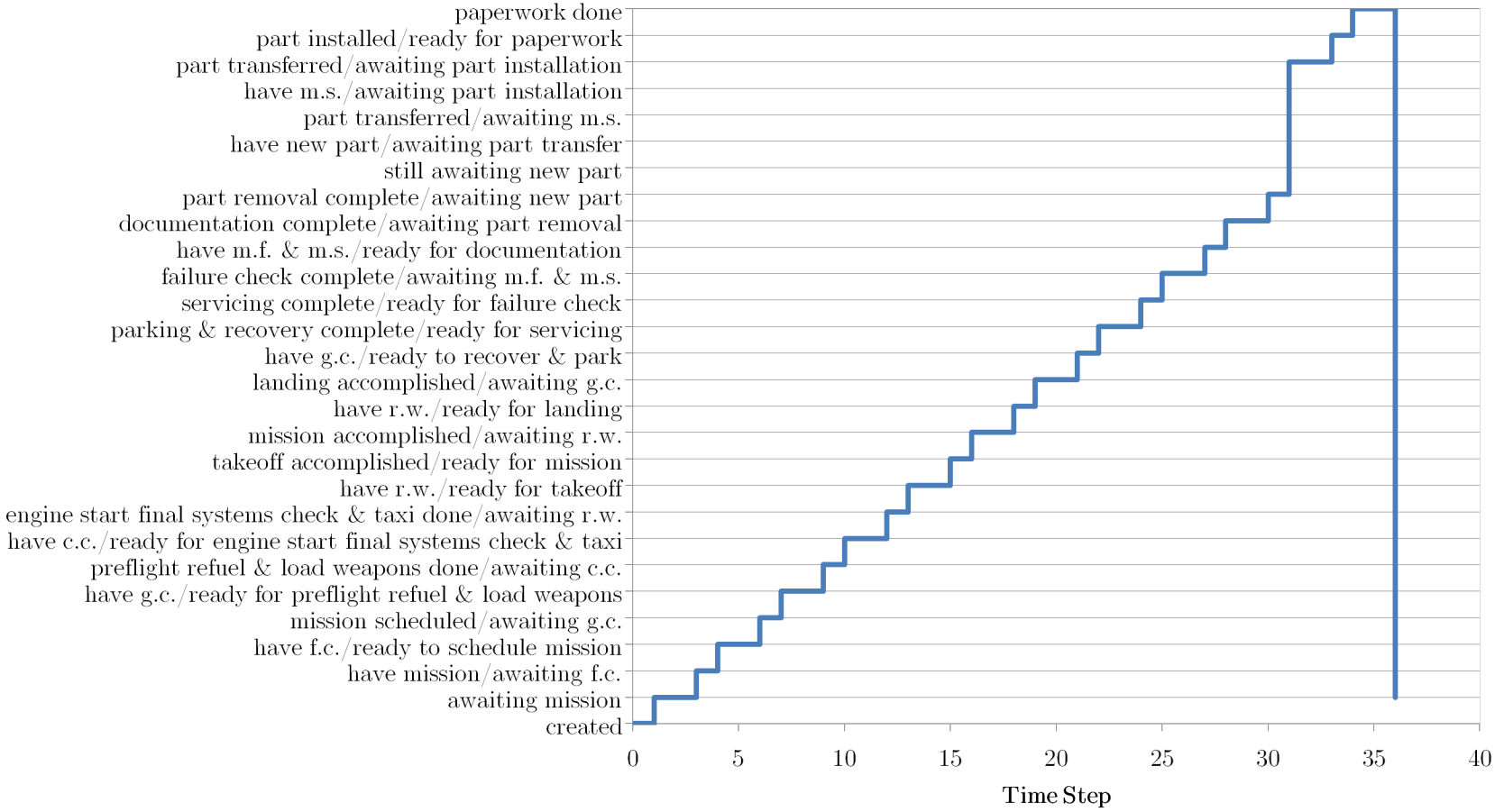


Figure 35: Path 3 event trace diagram

an available state. After some time, the aircraft is selected to fly another mission and the cycle repeats itself. This time, however, the delays associated with the ground crew are longer than they were the first time; this is because on the first cycle the aircraft was first in the queue for resources, while on subsequent missions a queue has formed.

Discussion has been made throughout this thesis of the concepts of uptime and downtime, which translate to the A_O metric. The uptime and downtime can be easily seen on Figure 36 as all downtime steps are grouped at the top of the figure (all steps past "servicing complete/ready for failure check") and all uptime steps are grouped at the bottom. This is shown more clearly in Figure 37, where those portions of the graph have been colored red, for downtime, and green, for uptime. In this way, the relative frequency of maintenance visits can be seen as the red portion is less populated than the green, indicating that failures do not occur frequently and, due to the overabundance of parts, aircraft leave rapidly once they require maintenance. This represents an ideal case, since maintenance *must* happen for the fleet to continue operating, but once it occurs it is desirable for it to be fast. Two of the decision points defined in Figure 32 are visible as the aircraft proceeds three times into the red region and four times does not. The third decision point is not in evidence since no aircraft ever wait for parts in this version of the model, as evidenced by the fact that several of the maintenance steps are skipped every time.

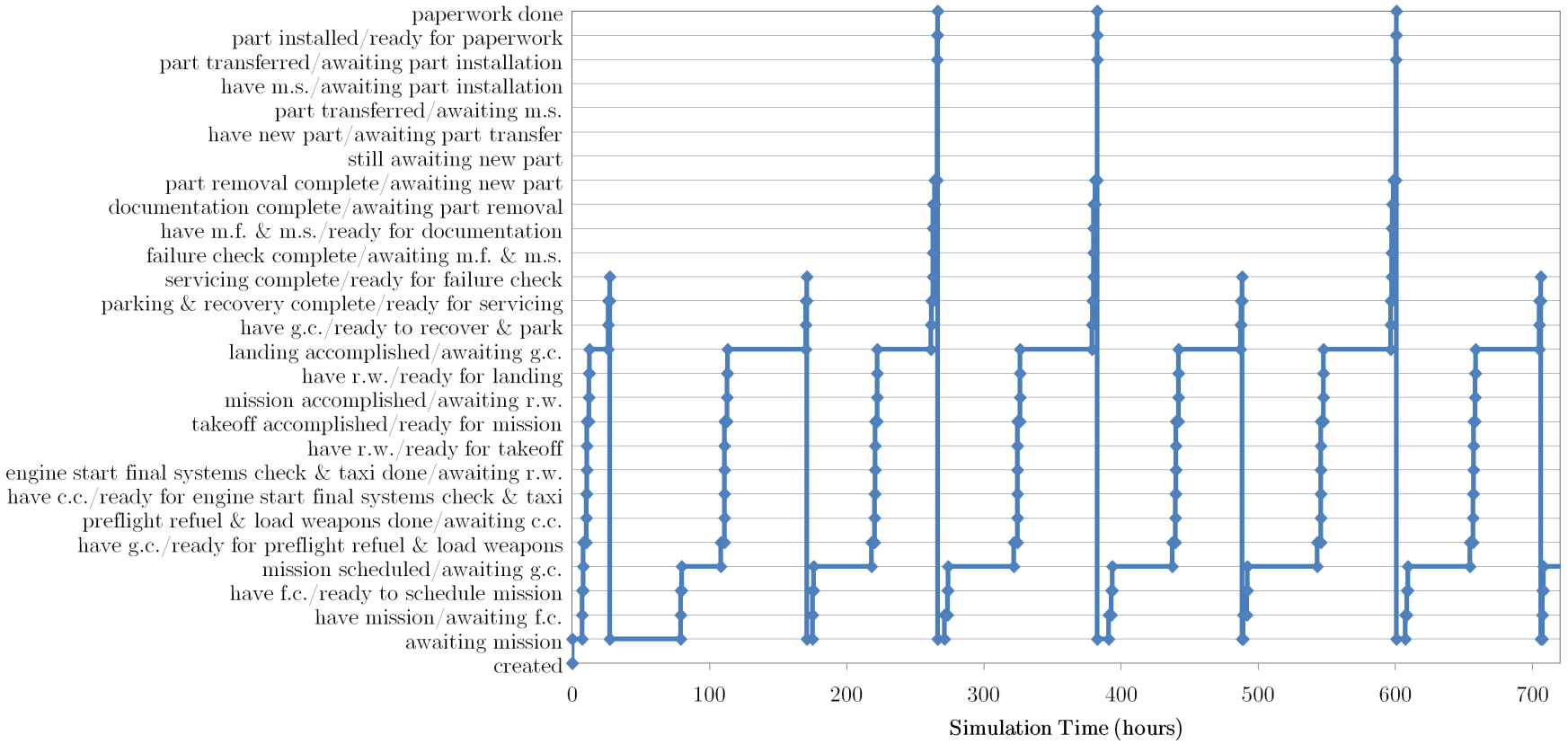


Figure 36: Aircraft event trace diagram, 30 days

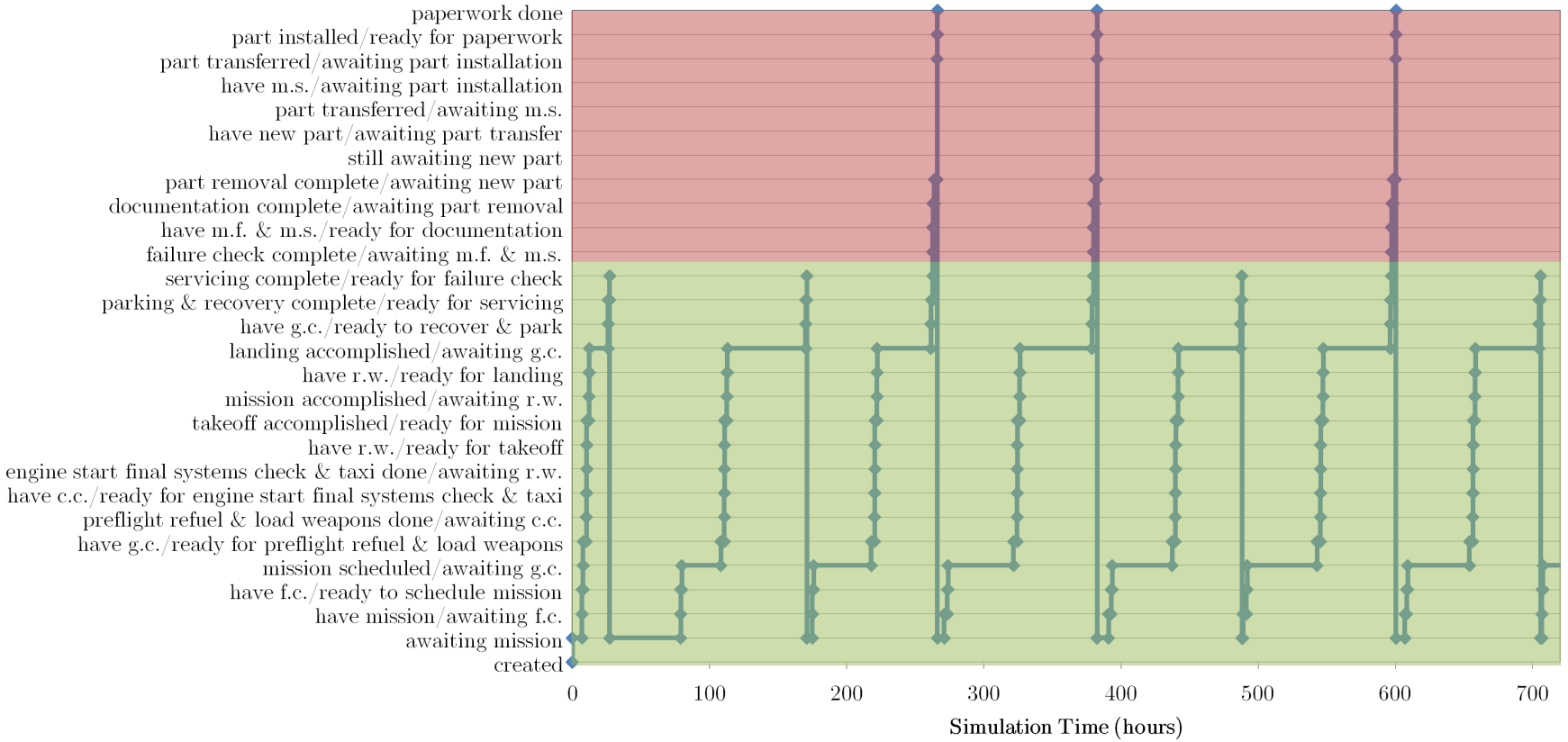


Figure 37: Aircraft event trace diagram, 30 days

Though Figures 36 and 37 have shown a small window of the operational period for an aircraft, this small slice of time will not necessarily reveal the whole behavior of the simulation. To gain this understanding, Figure 38 shows the event trace diagram for the full simulation time for a single aircraft. This plot shows that the aircraft operates for the full 365 days (8760 hours) of the simulation, indicating that nothing prevents it from carrying on its operations as expected. Furthermore, the spacing of maintenance visits within the red region of the plot indicates that the maintenance visits occur with stochastic spacing, as theorized in Chapter 1.

Though Figure 38 shows the event trace for a single aircraft, the remainder of the aircraft are not shown because their behavior looks essentially the same; though minor differences can be observed, they will not be noticeable and will overwhelm the plot. However, to show the overall behavior of the fleet the path frequencies have been computed for each of the individual aircraft as well as the fleet as a whole.

Table 9: Frequency and percent of path occurrence for fleet operations

	Path 1	Path 2	Path 3
Frequency	1771	609	0
Percent	0.74	0.26	0

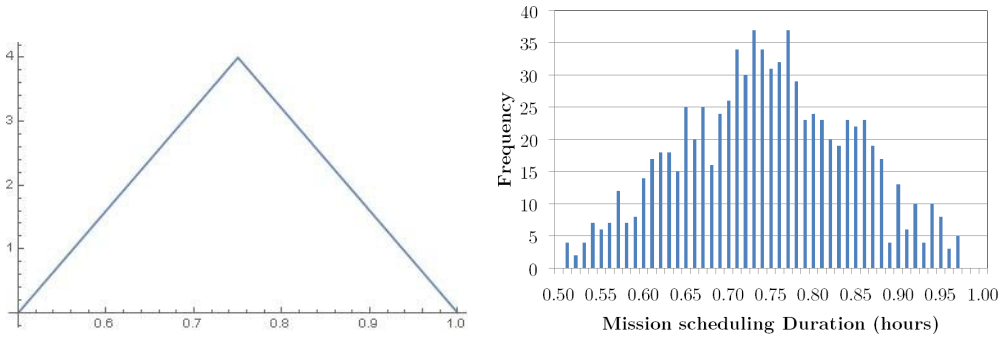
The aircraft had fairly similar results for the path frequency, with the path breakdowns varying from 80%/20% to 64%/36% splits across all the aircraft. Table 9 shows the number of times each of the different paths was followed by any of the aircraft in the fleet as well as the percentage of the time that any path was taken; as expected, the third path indicating that no parts are available is never experienced because by definition in this version of the model, parts are always available. Furthermore, the split between the first two paths is that Path 1 was taken about 75% of the time and Path 2 was taken about 25% of the time. Given that Sustain-ME at this point is represented only by the overall reliability of the aircraft, with failures every 6 flight hours, and that the most common sortie duration is 1.5 hours, it is expected that

aircraft fail every four missions on average ($\frac{6}{1.5} = 4$). This means that 25% of the time failures should occur, and this matches closely with the frequency that was observed, indicating that the modeling of aircraft failure is accurate.

Figure 38: Aircraft event trace diagram, 365 days

4.5.2 Distribution Verification

Having confirmed that the order of events in Sustain-ME follows the expected order, now the time distributions for the duration of different steps can be examined to confirm this aspect of Sustain-ME. These distributions are shown in Figures 39 through 52 as a comparison between the theoretical distribution and the observed values from Sustain-ME. They indicate that the model correctly generates random step durations from the distributions as desired. Though most of the histograms were generated from about 800 observations collected from the model, the mission scheduling histogram was replotted using about 100,000 observations to show that the limiting case of these distributions does closely resemble a triangular distribution. The more granular histogram is provided in Figure 40. Table 10 further compares the minimum, maximum, and mode of the observed values to the parameters of the theoretical distributions to show close agreement between the two.



(a) Theoretical (b) Observed

Figure 39: Comparison of distributions for mission scheduling step

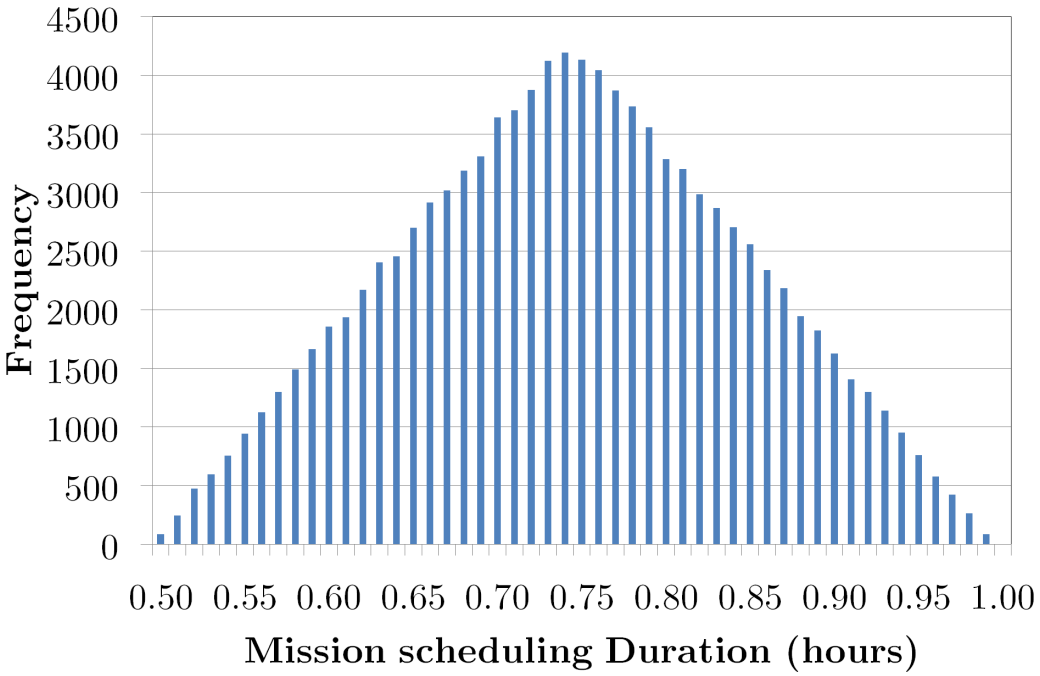
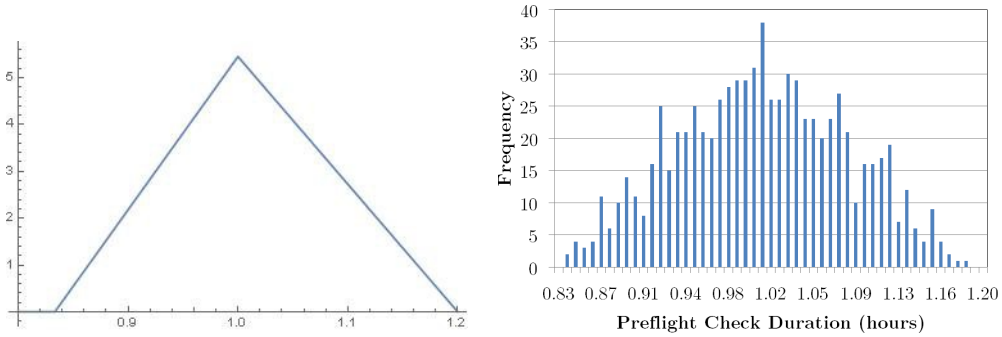
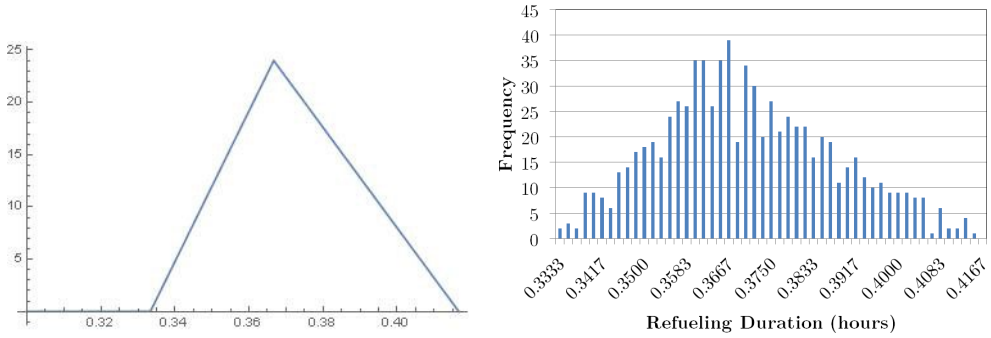


Figure 40: High granularity histogram



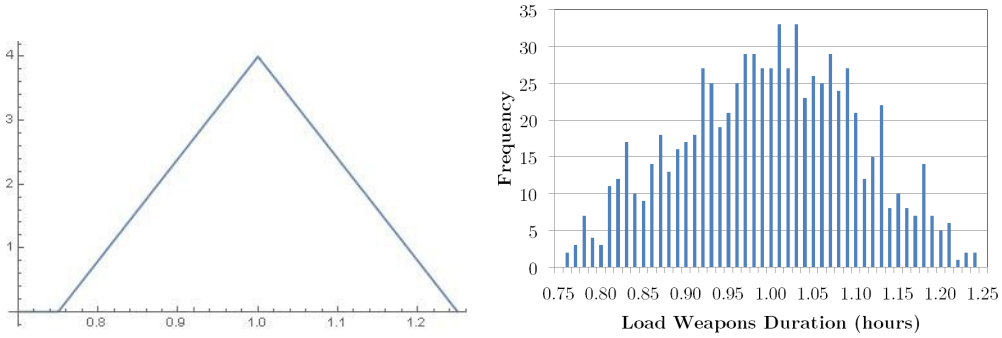
(a) Theoretical (b) Observed

Figure 41: Comparison of distributions for preflight check step



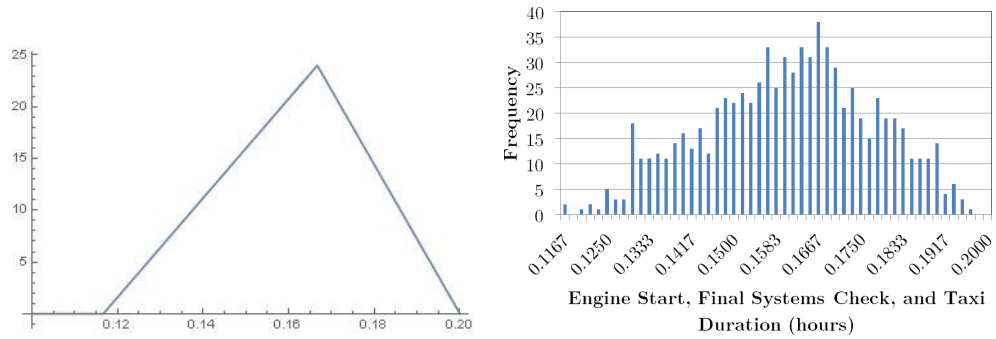
(a) Theoretical (b) Observed

Figure 42: Comparison of distributions for refueling step



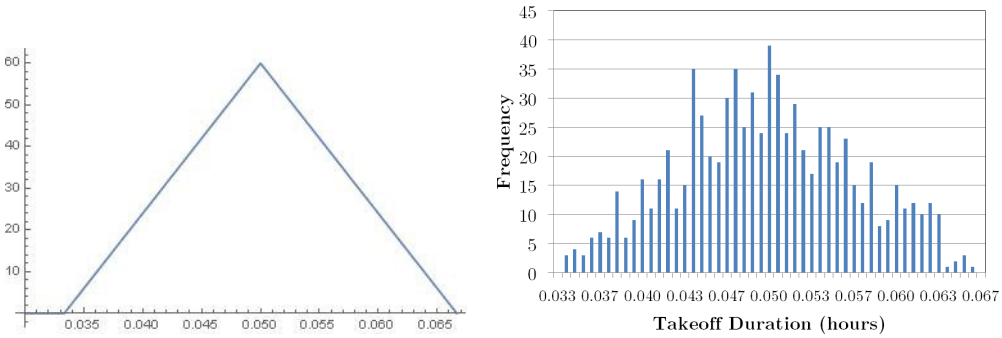
(a) Theoretical (b) Observed

Figure 43: Comparison of distributions for load weapons step



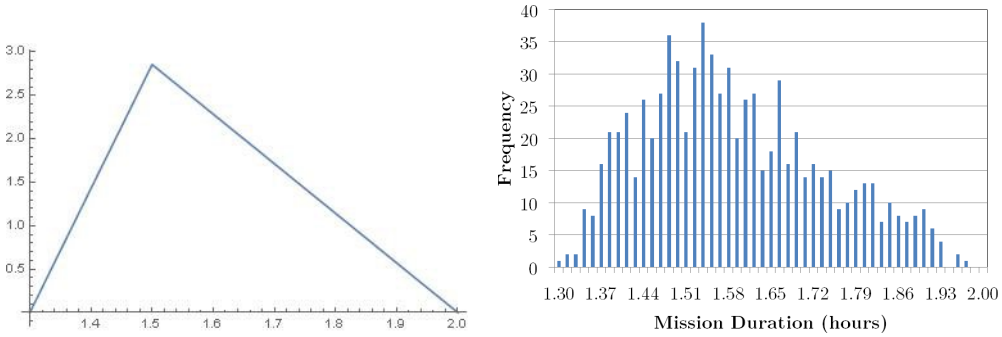
(a) Theoretical (b) Observed

Figure 44: Comparison of distributions for engine start, final weapons check and taxi step



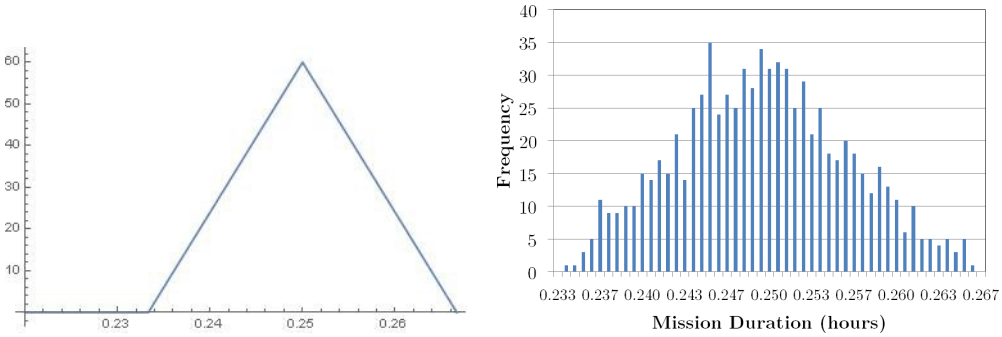
(a) Theoretical (b) Observed

Figure 45: Comparison of distributions for takeoff step



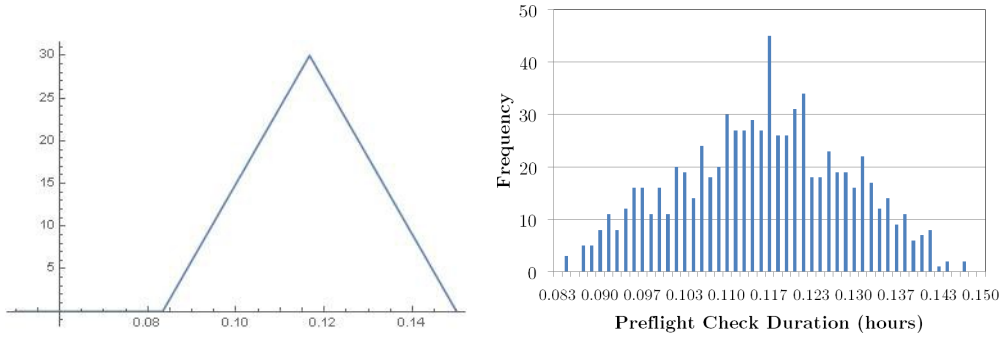
(a) Theoretical (b) Observed

Figure 46: Comparison of distributions for fly mission step



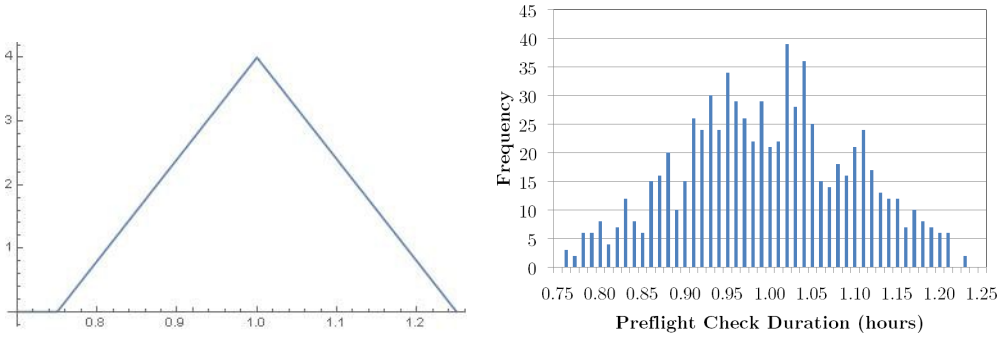
(a) Theoretical (b) Observed

Figure 47: Comparison of distributions for landing step



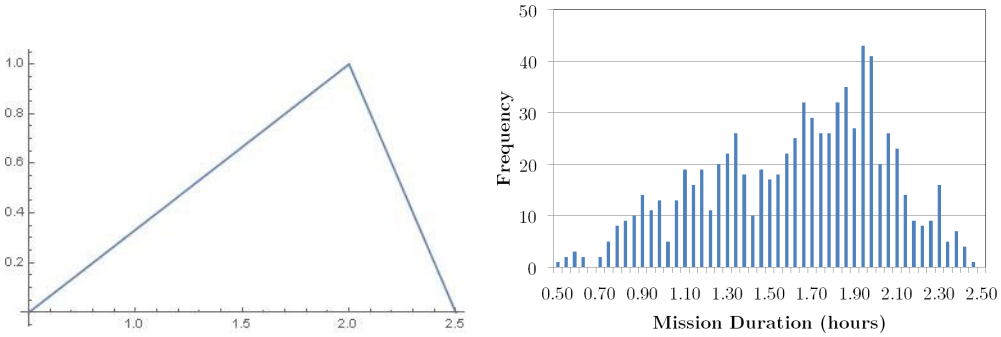
(a) Theoretical (b) Observed

Figure 48: Comparison of distributions for parking and recovery step



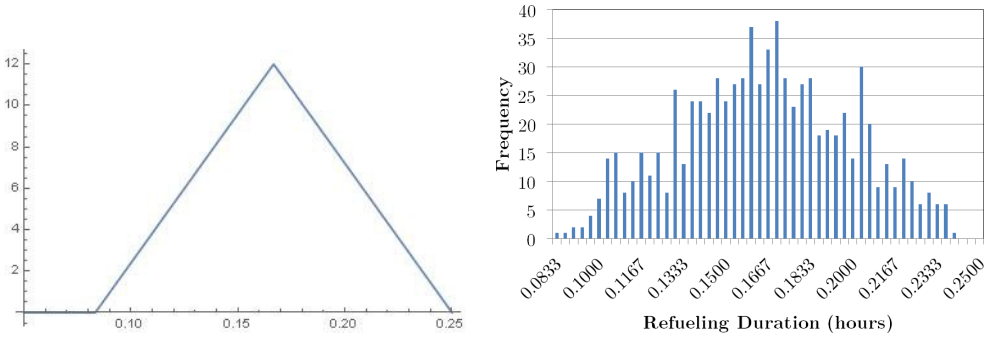
(a) Theoretical (b) Observed

Figure 49: Comparison of distributions for servicing step



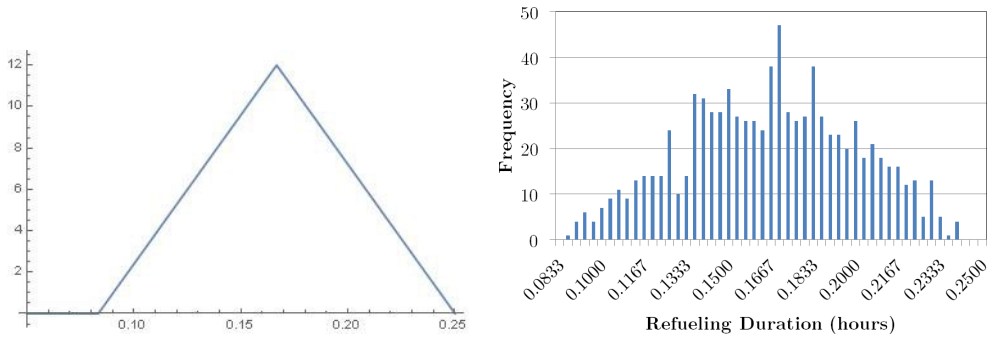
(a) Theoretical (b) Observed

Figure 50: Comparison of await inventory step



(a) Theoretical (b) Observed

Figure 51: Comparison of paperwork step



(a) Theoretical (b) Observed

Figure 52: Comparison of documentation step

Table 10: Theoretical and observed parameters of simulation step distributions

Step		Min	Mode	Max
Mission Scheduling	Theoretical	0.5	0.75	1.0
	Actual	0.5109	0.75	0.9790
Preflight Check	Theoretical	$0.8\overline{3}$	1.0	1.2
	Actual	0.8443	1.009	1.1876
Refueling	Theoretical	$0.\overline{3}$	$0.3\overline{6}$	$0.41\overline{6}$
	Actual	0.3343	$0.3\overline{6}$	0.4152
Load Weapons	Theoretical	0.75	1.0	1.25
	Actual	0.7609	1.02	1.2473
Engine Start, Final Weapons Check and Taxi	Theoretical	$0.11\overline{6}$	$0.1\overline{6}$	0.2
	Actual	0.1178	$0.1\overline{6}$	0.1968
Takeoff	Theoretical	$0.0\overline{3}$	0.05	$0.0\overline{6}$
	Actual	0.03403	0.05	0.06648
Fly Mission	Theoretical	1.3	1.5	2.0
	Actual	1.3067	1.538	1.9850
Landing	Theoretical	$0.2\overline{3}$	0.25	$0.2\overline{6}$
	Actual	0.2346	0.2453	0.2664
Parking and Recovery	Theoretical	$0.08\overline{3}$	$0.11\overline{6}$	0.15
	Actual	0.08569	$0.11\overline{6}$	0.1485
Servicing	Theoretical	0.75	1.0	1.25
	Actual	0.7680	1.02	1.2314
Await Inventory	Theoretical	0.5	2.0	2.5
	Actual	0.5249	1.94	2.4666
Paperwork	Theoretical	$0.08\overline{3}$	$0.1\overline{6}$	0.25
	Actual	0.08447	0.17	0.2404
Documentation	Theoretical	$0.08\overline{3}$	$0.1\overline{6}$	0.25
	Actual	0.08819	0.17	0.2430

4.5.3 Operational Availability

One final verification activity needs to be performed. Since the behavior of the fleet at this point was modeled with a simple supply chain that is equivalent to a level of resources that is more than adequate to support the fleet, the operational availability of Sustain-ME should be checked to determine that the A_O is, in fact, high. But more importantly, this will provide a point of comparison for the model with the supply chain implemented, as providing that version of Sustain-ME with more than adequate resources should show similar behavior. Figure 53 shows that the A_O of the fleet with a highly responsive supply chain is very good (essentially bounded between an A_O of 1 and 0.95) and remains so over the course of the simulation (only the first 60 days are shown to avoid compressing the stochasticity in the horizontal dimension). This figure shows ten reptitions of Sustain-ME, which indicates that there is stochasticity in the behavior of Sustain-ME, but that it does not affect the overall trend. This indicates that, as expected, the behavior of sustainment is nonstationary at high inventory levels.

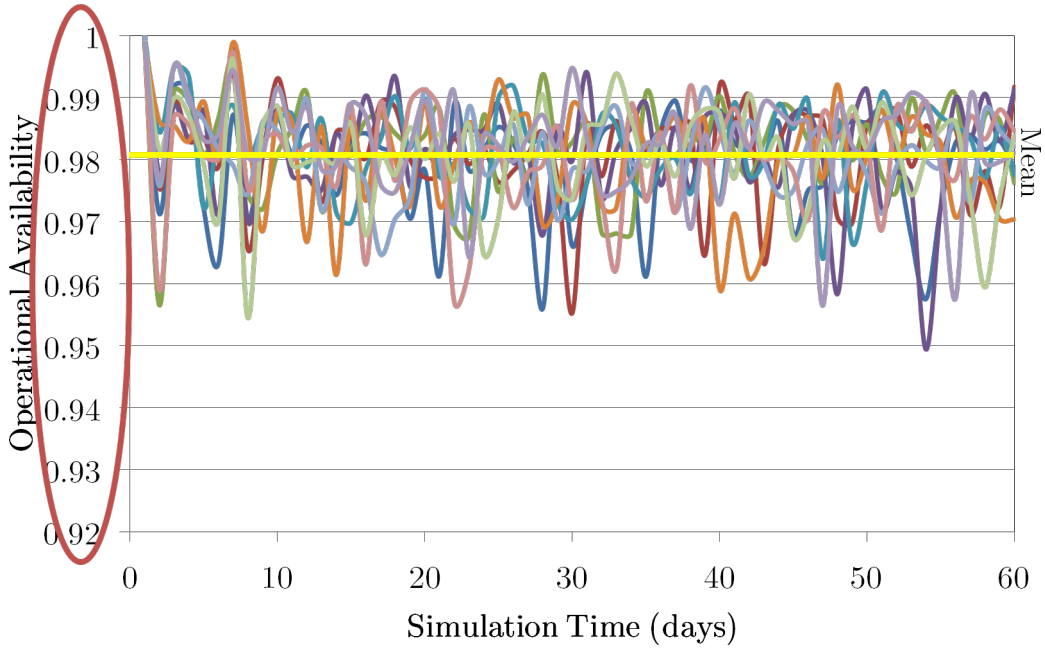


Figure 53: Operational availability with minimum resources

4.5.4 Fleet Operations Excluding Supply Chain Conclusions

Having now explored the behavior of the fleet operations before implementing the supply chain, the fleet is found to carry out different activities with the frequency and duration matching the assumed values given to Sustain-ME. The events of the model also match those that were shown in Figure 32, and the aircraft were found to operate for the full simulation without getting stopped at any step. Having seen the results of these tests, confidence in the fleet operations portion of Sustain-ME is sustained. At this point the supply chain behavior can be integrated, as is verified in Section 4.6.

4.6 Fleet Operations Including Supply Chain: Verification

This section covers the modules denoted by the letter 'D' in Figure 22. The supply chain was modeled based on the distributions and ordering logic described in Section 3.1.4. Because it opens up the possibility of conditions within Sustain-ME where not enough spare parts are available, there are several tests that should be performed to verify the behavior of the model. The first of these should plot the operational availability over time with a high number of spare parts and compare this plot to the results from Section 4.5, since this model represents the limiting case of Sustain-ME with the supply chain included. If the results differ significantly, this suggests that some error has been introduced in the new model version since the supply chain logic would not impact the fleet's performance at conditions of excess inventory.

The next test that should be performed is to plot the operational availability versus time for a wide range of inventory levels. Logically, decreasing inventory should lead to a degradation in the operational availability of the fleet as aircraft are forced to wait longer periods for replacement parts to become available. However, this result should be observed over a medium range of inventory as was theorized in Chapter 2. Above this range, the operational availability should be consistently high, with additional

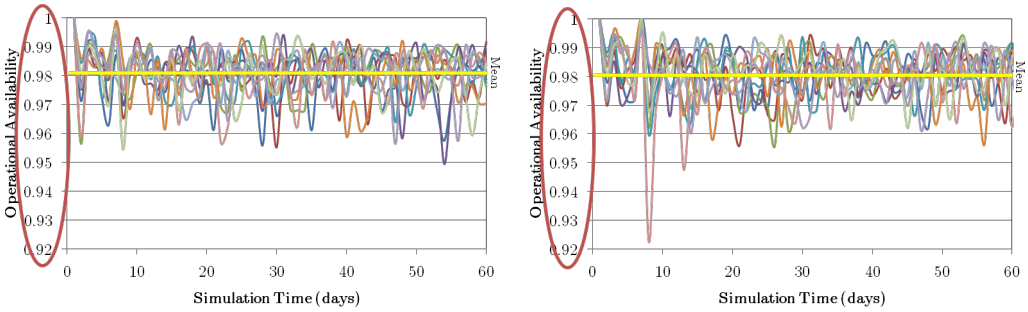
spare parts not contributing any further increase in operational availability because parts are always available when needed. Below this range, the operational availability should be consistently low, with a decrease in spare parts not significantly impacting the operational availability because the operational availabilty is already so poor as to be effectively zero. In the second case, however, the simulation is expected to show an initial period of high operational availability before any part failures have occurred, during which time the fleet can operate without difficulty because the initial inventory investment has not yet been used up.

The next test should duplicate one performed in Section 4.5 as well, plotting the state of each of the aircraft over time to visually inspect the behavior for anomalies, and additionally checking the behavior by comparing event sequences to known paths. However, with the inclusion of parts as objects within the code, a similar test should be performed to visually inspect the behavior of the parts objects over time to ensure their adherence to intended rules. The specific way in which parts will be visually examined will be described in detail later. The final test for this portion of the code should implement multiple parts on the aircraft, since the introduction of more than one part category could introduce error.

4.6.1 Supply Chain with High Inventory Level Comparison

Since the first test of the supply chain logic is to compare the behavior of Sustain-ME to the previous version of Sustain-ME where parts were assumed to be immediately available, any inventory level which is sufficient to ensure that parts are always on hand should match the behavior demonstrated in Figure 53. To make certain that parts were always available, the spare parts inventory was set at 50,000. However, as will be shown in a future test, a much smaller number of parts is sufficient to achieve this result. Figure 54 shows that the operational availability of the supply chain model with 50,000 parts matches that of the fleet model excluding the supply

chain. Furthermore, Table 11 shows that the difference between the time averaged operational availability across ten repetitions is negligible. Taken together, these indicate that no error has been introduced by the portion of the supply chain code that sends parts from local inventory to aircraft, or by the creation of parts objects as a mechanism for allowing aircraft repair.



(a) Fleet Model Excluding Supply Chain

(b) Fleet Model Including Supply Chain, 50,000 Parts

Figure 54: Interdependence of sequential queues

Table 11: Time averaged operational availability comparison

Model Excluding Supply Chain	0.9813
Model Including Supply Chain	0.9810
Percent Difference	0.03%

4.6.2 Effect of Inventory on Operational Availability

The next test of the supply chain logic and coding looks at the operational availability behavior of Sustain-ME over a range of inventory levels to ensure that decreasing inventory decreases fleet performance as expected. Figures 55 through 58 show that the behavior at different levels of inventory does in fact decrease as the inventory is decreased. Note that the behavior of Sustain-ME for 250 spare parts looks very similar to the behavior of Sustain-ME for 50,000 parts, since both represent cases where spares were always available when needed. Also note that the behavior of the model for zero spare parts does not exhibit 0% uptime as might be expected, but rather shows periods of slight recovery. This occurs because, while no *spare* parts were

created in Sustain-ME, the parts removed from the aircraft are eventually repaired by the depot and returned to service. This allows for aircraft to be repaired, but over a time frame limited by the length of the depot repair cycle.

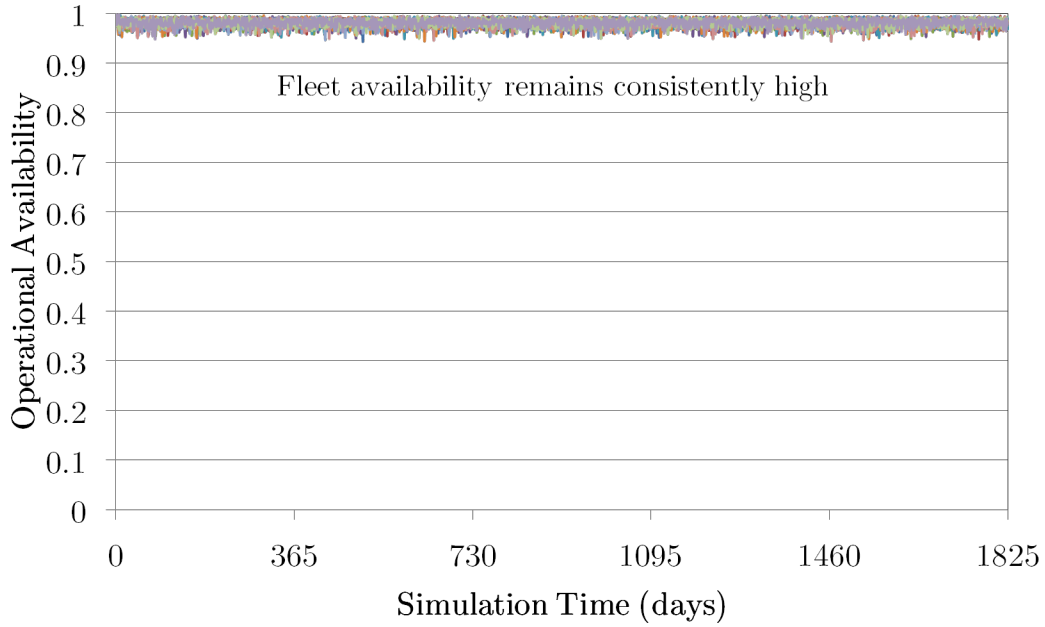


Figure 55: Model behavior with 250 spare parts

A few other observations can be made from this data. The first is illustrated in Figure 59, which plots the inventory level versus the time averaged operational availability achieved in ten different repetitions of Sustain-ME (an amount that was selected based on observing that the mean across repetitions does not change after about 5 repetitions, so the mean across reptitions can be confidently stated after ten repetitions). The inventory level is shown on a log scale due to the fact that the operational availability changes significantly over a narrow band of inventory levels. In this region, a small change in the spare inventory provided yields a large change in the average operational availability. Unsurprisingly, this region is also characterized by larger variability in the operational availability both within a single model run as well as between model runs. Figure 59 showed the variability between runs, since at medium inventory levels the spread of the average A_O over time for ten different simulations varied widely. The variability *within* runs is best demonstrated

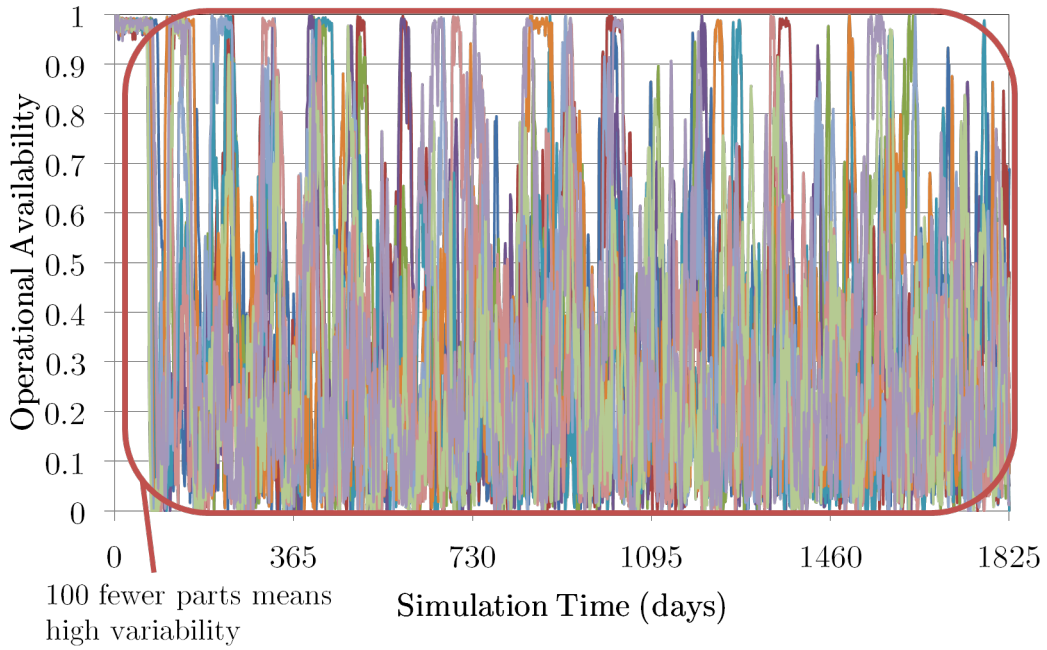


Figure 56: Model behavior with 150 spare parts

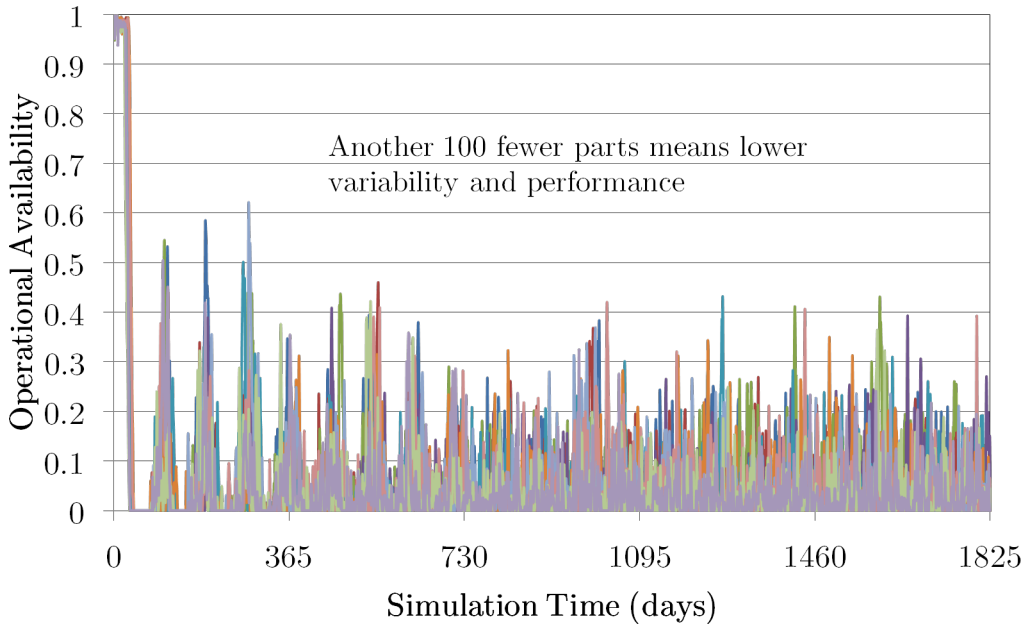


Figure 57: Model behavior with 50 spare parts

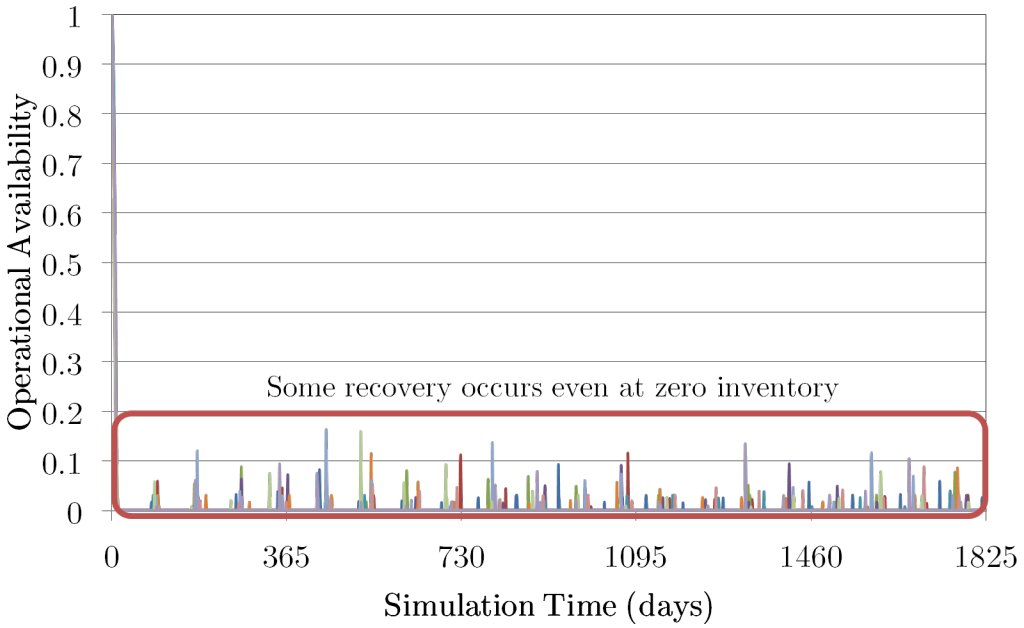


Figure 58: Model behavior with 0 spare parts

in Figure 56, which shows the variability in operational availability from day to day (a single line on the chart) as well as from repetition to repetition (the difference between the lines). This high degree of variability is the same behavior predicted in Chapter 1 and in the literature. Outside this region of rapid change, Sustain-ME's behavior becomes closer and closer to the limiting cases discussed in Chapter 1, where queue lengths go to zero (at the high end) or queue lengths go to their maximum (at the low end).

Taken together, these results show that Sustain-ME follows the expected behavior as the inventory is reduced. Moreover, this behavior emerges not from directly setting up the model to field fewer aircraft when there are fewer spare parts, but from setting up a supply chain with basic but realistic rules and seeing how it reacts at different inventory settings. Though this is not enough on its own to verify that the model is behaving as desired, it does suggest that some degree of confidence in the effect of the supply chain can be maitained.

4.6.3 Event Activity Verification

The next test of the supply chain model reexamines the states that each of Sustain-ME's aircraft moves through over time. In order to ensure that behaviors which come into play at different inventory levels are not missed, plots will be made of the state of the model's aircraft at inventory levels from the three regions of Figure 59: high average A_O, medium average A_O, and low average A_O. Representing these regions are, respectively, 225 parts, 180 parts, and 25 parts.

Figures 60 and 61 shows the behavior of the fleet with the supply chain implemented and with 225 spare parts, within the region where parts are always available. The figure looks remarkably similar to Figure 38 and Figures 36 and 37, and this makes sense because the behavior of the two models should be fairly similar at high inventory conditions. Just like in Section 4.5.1, the path frequency data reveals that the first two paths in Figure 32 are visited while the third is not, since the fleet never has to wait for parts. This is shown in Table 12, where the percentages are still very close to 75% for Path 1 and 25% for Path 2.

Table 12: Frequency and percent of path occurrence for fleet operations

	Path 1	Path 2	Path 3
Frequency	1727	656	0
Percent	0.72	0.28	0

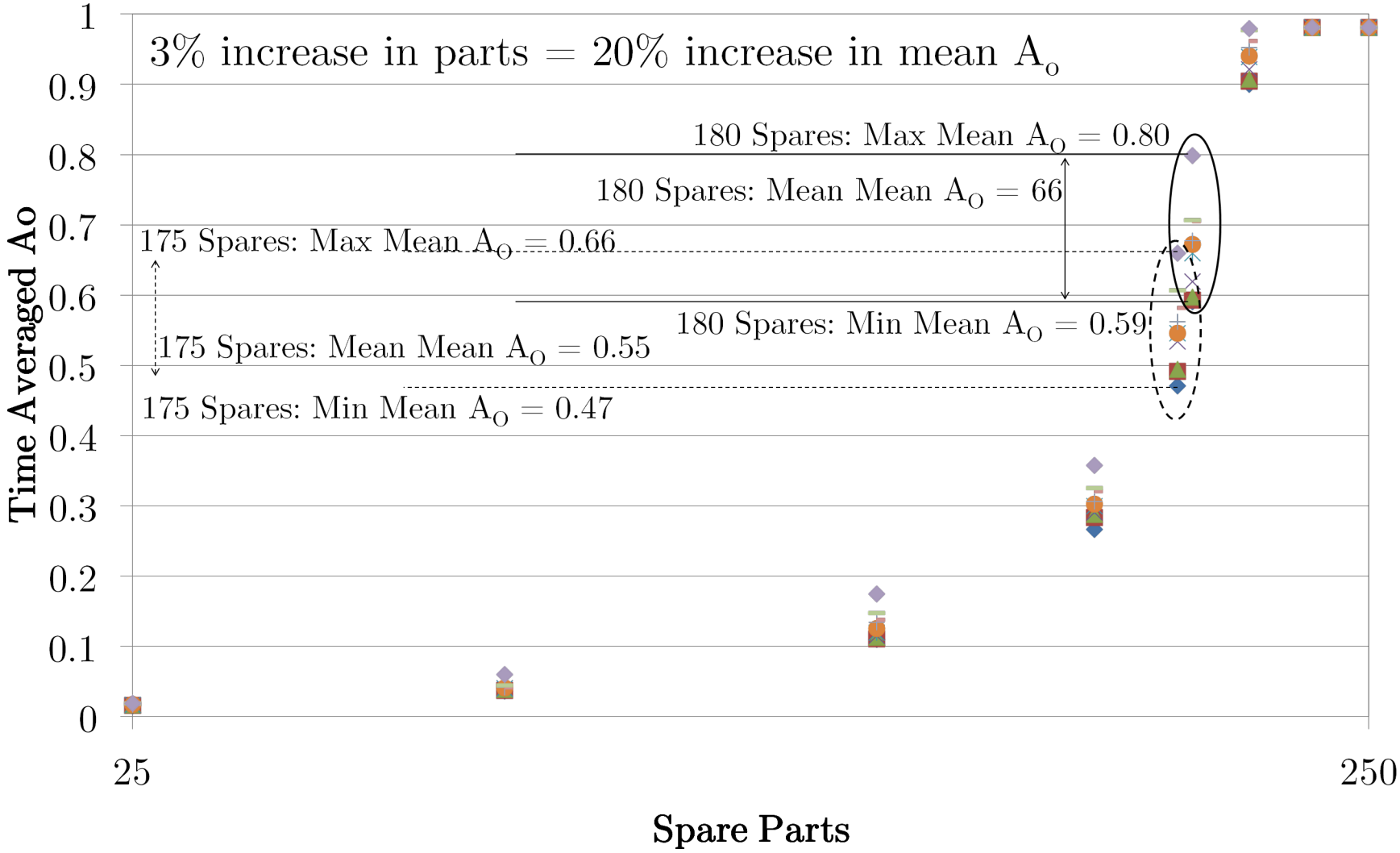


Figure 59: Time averaged A_O versus inventory

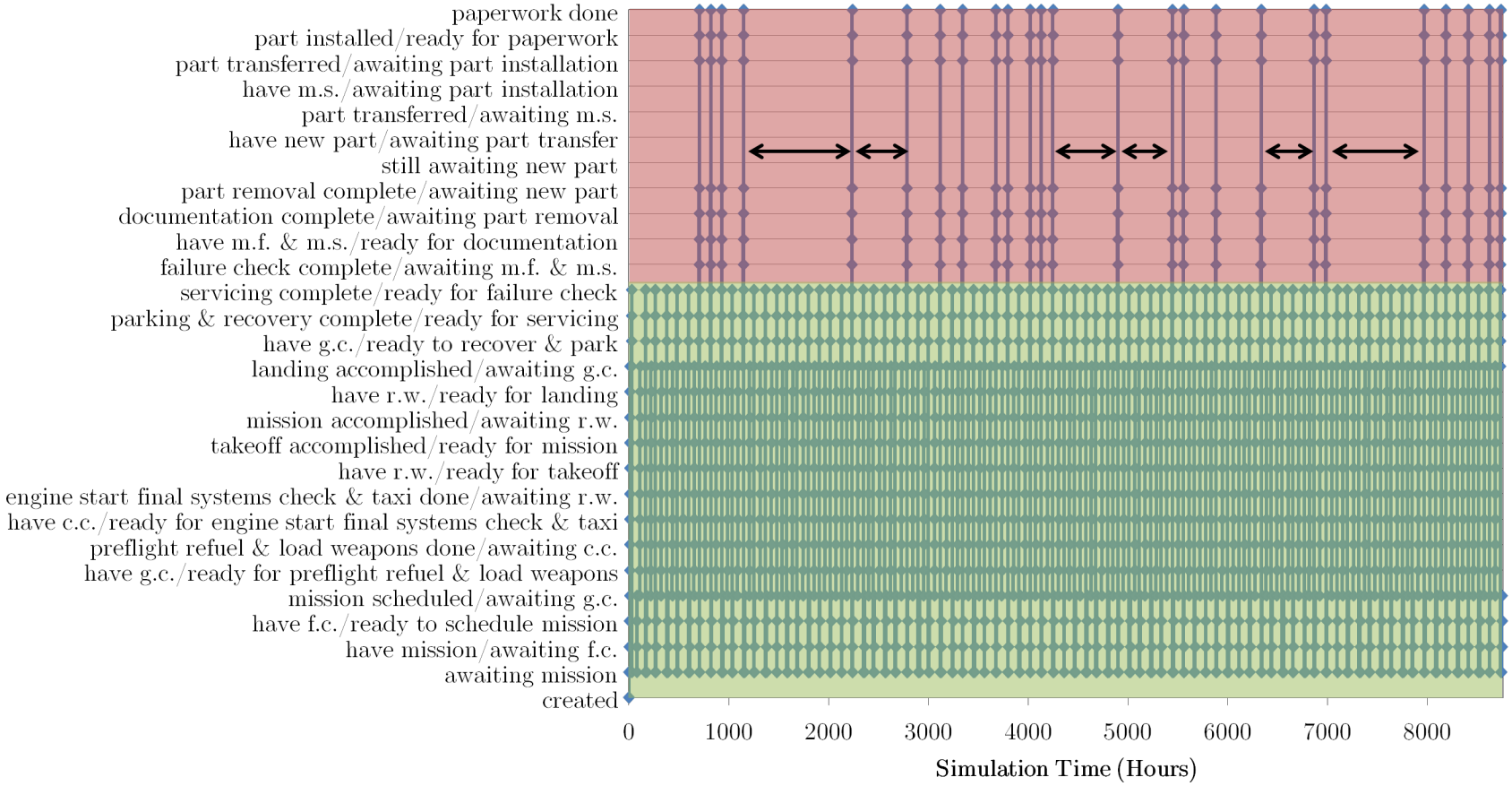


Figure 60: Event trace diagram for 225 spare parts, 365 days

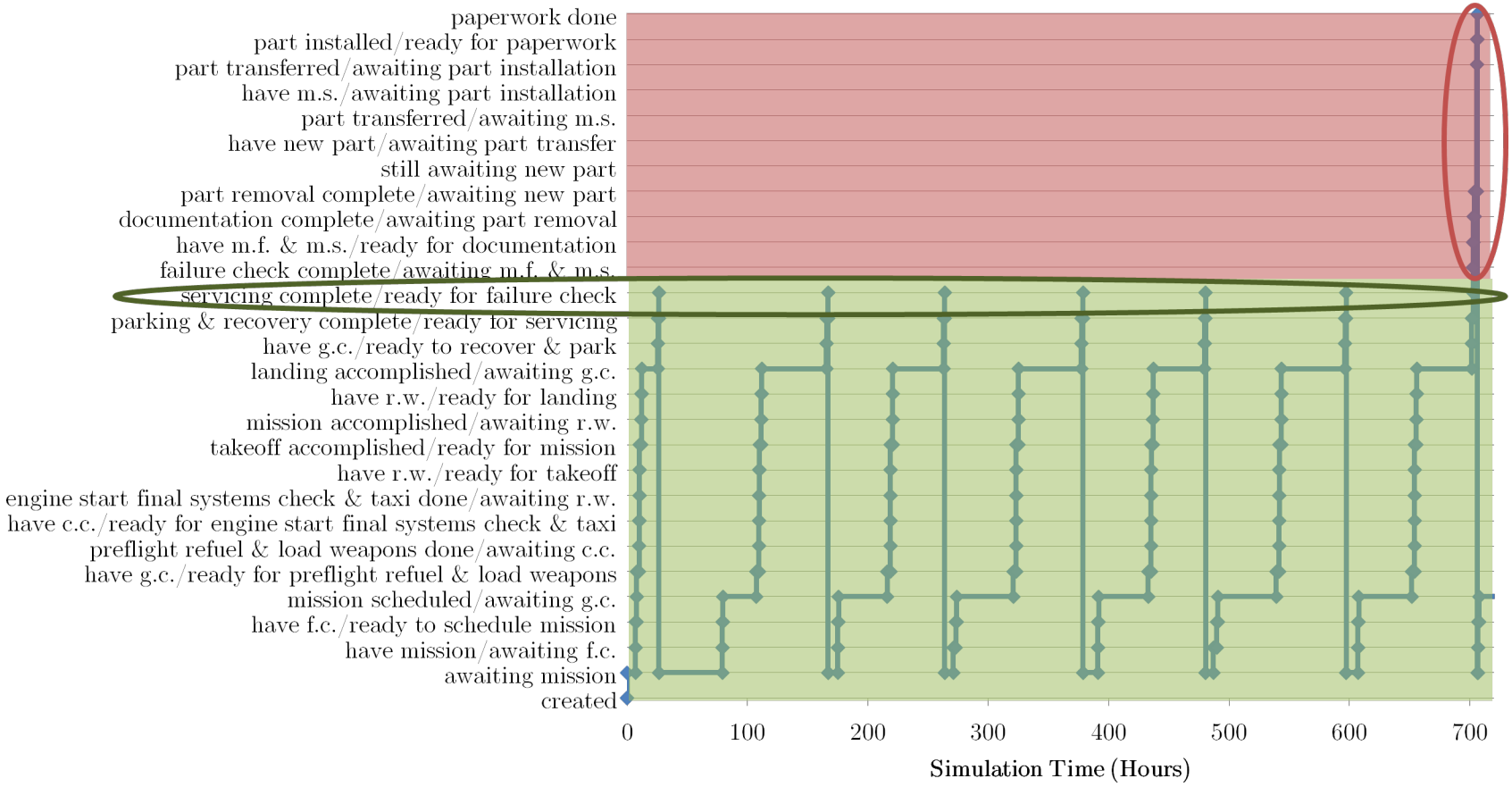


Figure 61: Event trace diagram for 225 spare parts, 30 days

Figure 62 shows the events of the simulation when the initial inventory is 180 spare parts, which is within the region of high variability shown in Figure 59. Where the behavior was constant throughout time in Figure 60, indicating that the behavior is nonstationary, in this case significant delays can be seen as large horizontal gaps in the results of Figure 62. These gaps appear later in the simulation, rather than occurring throughout; this indicates that it takes time for the aircraft represented to "see" the effects of not having enough inventory. The other aircraft in the fleet exhibited similar behavior, though these results are not shown for brevity. Figure 63 shows a closer view of the time from 6000 simulation hours to the end of the simulation. This closer image shows that the aircraft goes through cycles of long delays, when it requires maintenance, and short cycles when it is able to fly without being maintained. The growing length of delays also indicates that the time length for which the simulation was run may not be sufficient to reveal the full behavior. Since most of the simulation time is spent in the same way with a new development near the end, the same 180 initial inventory case was run for a longer period of 5 years of simulation time, shown in Figure 64 without the states listed along the y-axis to keep as much horizontal space available as possible. Though it is difficult to see because of the length of time displayed, the white space in the bottom portion of Figure 64 indicates gaps that occur due to a wait for spare parts. The figure does not display a repeating pattern, but rather continues to display periods of delay that are both longer and shorter, and randomly spaced throughout time. This helps to indicate that the nonstationarity observed in Figure 56 is not a subset of a broader stationary pattern, but is truly random. The correspondence between these periods of delays and the spare parts that are on site will be shown in Section 4.6.4.

The test for path frequency shows that Path 3 is finally being visited now that aircraft occasionally have to wait for parts. Table 13 shows that the probability of seeing a failure remains the same, but is now split into two different probabilities

corresponding to Paths 2 and 3. Checking these probabilities against a computed value is more difficult since the amount of time that parts are not on hand is not a simple computation. However, it is expected that the frequency of Path 3 will go up as the initial inventory investment decreases.

Table 13: Frequency and percent of path occurrence for fleet operations

	Path 1	Path 2	Path 3
Frequency	1653	356	288
Percent	0.72	0.15	0.13

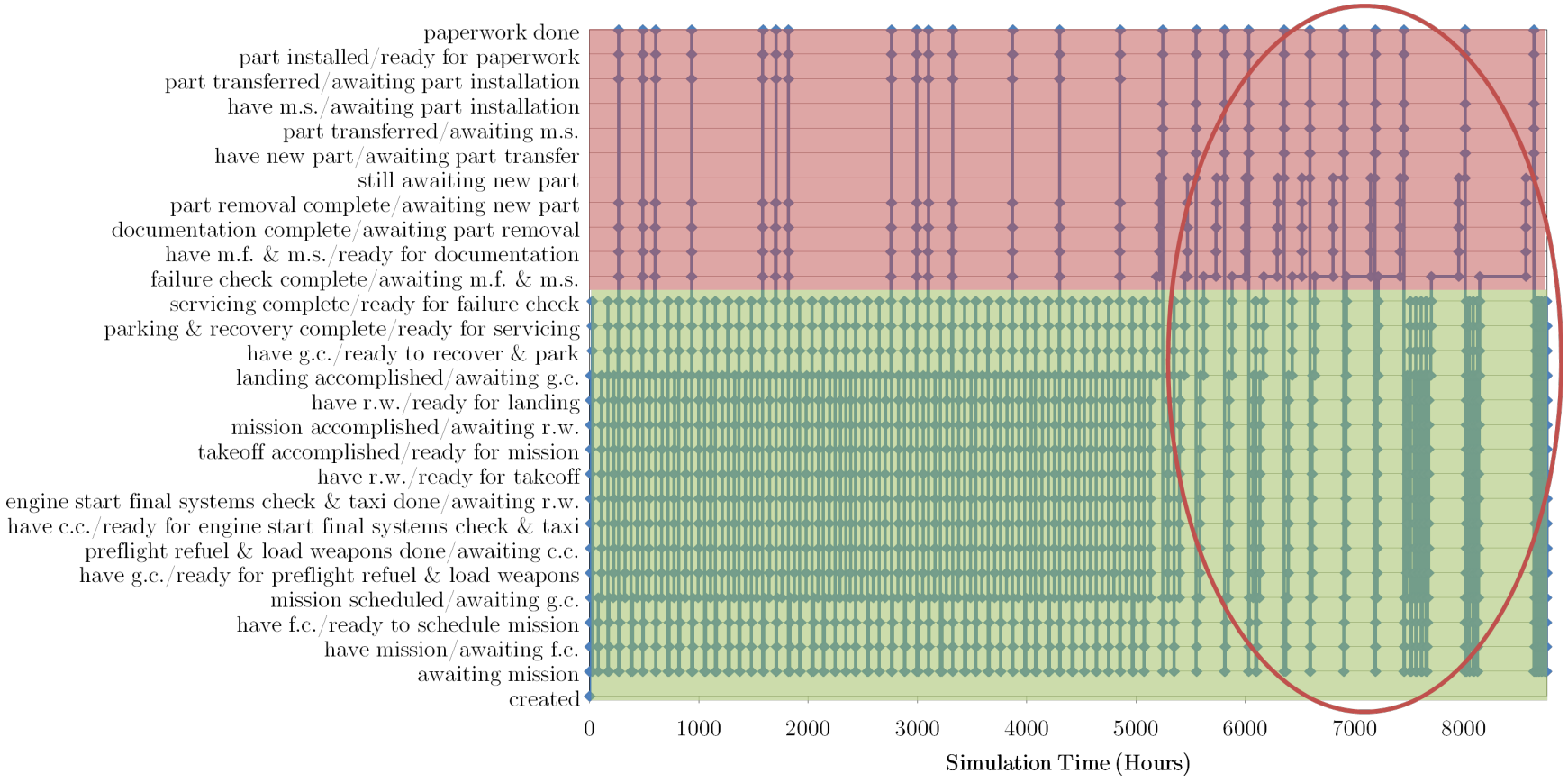


Figure 62: Event trace diagram for 180 spare parts, 365 days

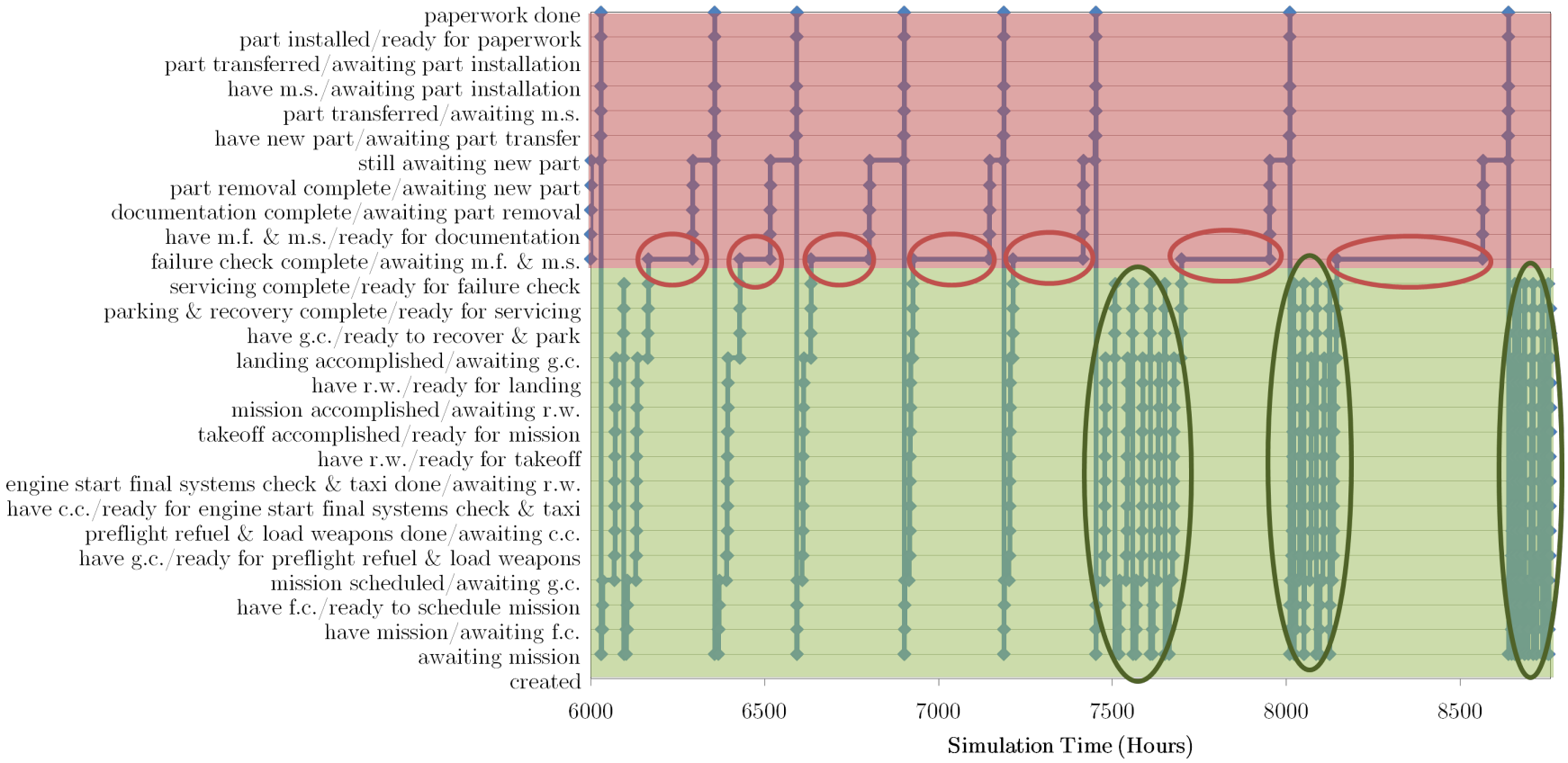


Figure 63: Event trace diagram for 180 spare parts, last 6000 hours

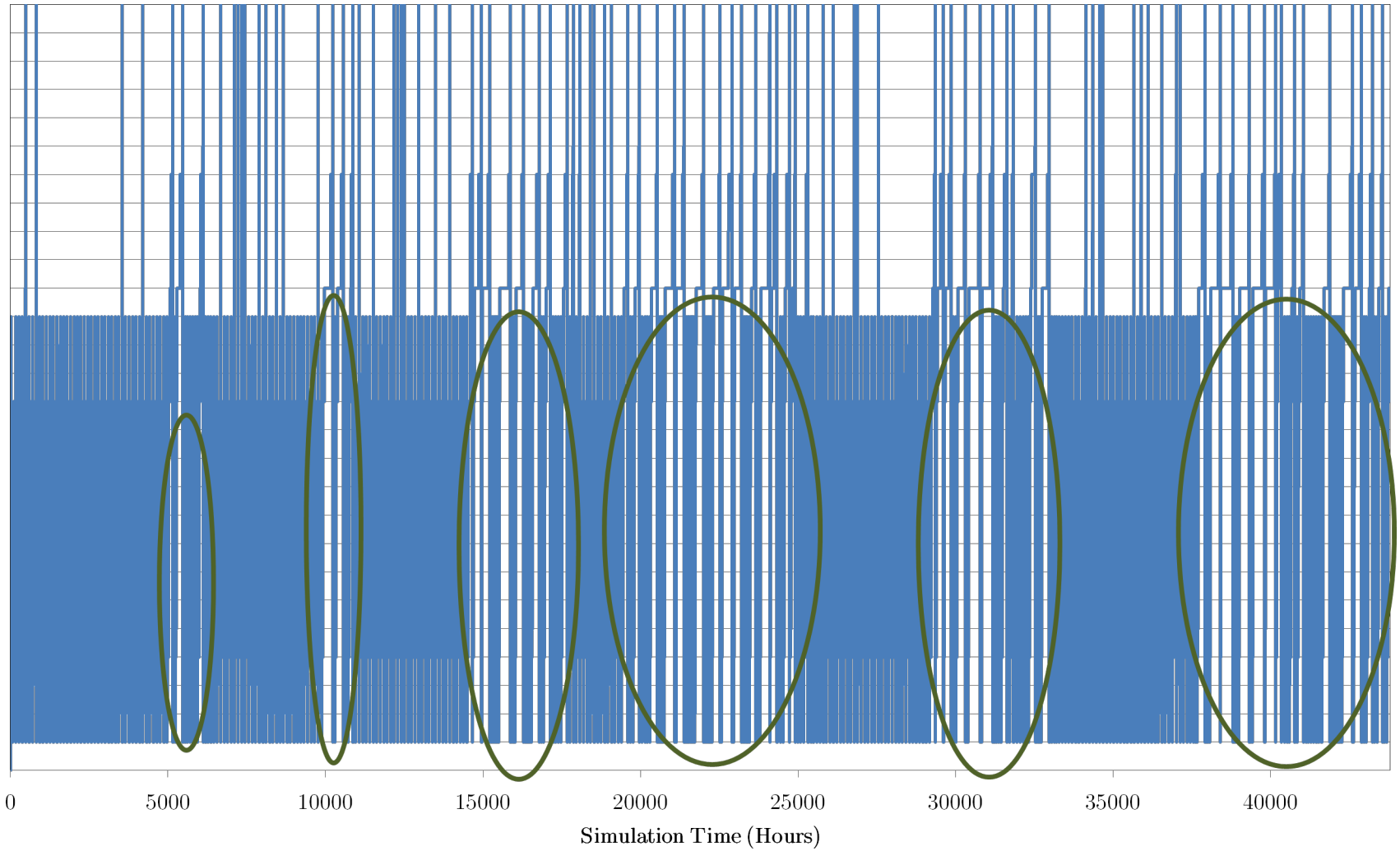


Figure 64: Event trace diagram for 180 spare parts, 5 years

The final aircraft behavior check is for the case where only 25 spare parts are initially available, which is in the low performing region of Figure 59. Figure 65 shows the event trace for an aircraft over the course of a year. It is characterized by a single maintenance loop without delays, after which every maintenance visit leads to delays on the order of eighty days. This amount is close to the refurbishment cycle length for spare parts, indicating that the consistent but low behavior is due to the aircraft almost always having to wait for parts to be refurbished and sent back before they can be again equipped on the aircraft.

Table 14 shows how the prediction made from Table 13 does in fact occur. First of all, due to the delays for spare parts the overall number of operational cycles (missions) for each aircraft is much lower than for 225 or 180 spare parts. However, the split between Paths 2 and 3 can be seen to fall much more heavily toward Path 3, as predicted from the data for 180 spare parts. This indicates that, most of the time, aircraft are required to wait for parts before returning to available status. Furthermore, Figure 65 shows that when they wait, they wait for a long time.

Table 14: Frequency and percent of path occurrence for fleet operations

	Path 1	Path 2	Path 3
Frequency	344	21	99
Percent	0.74	0.04	0.21

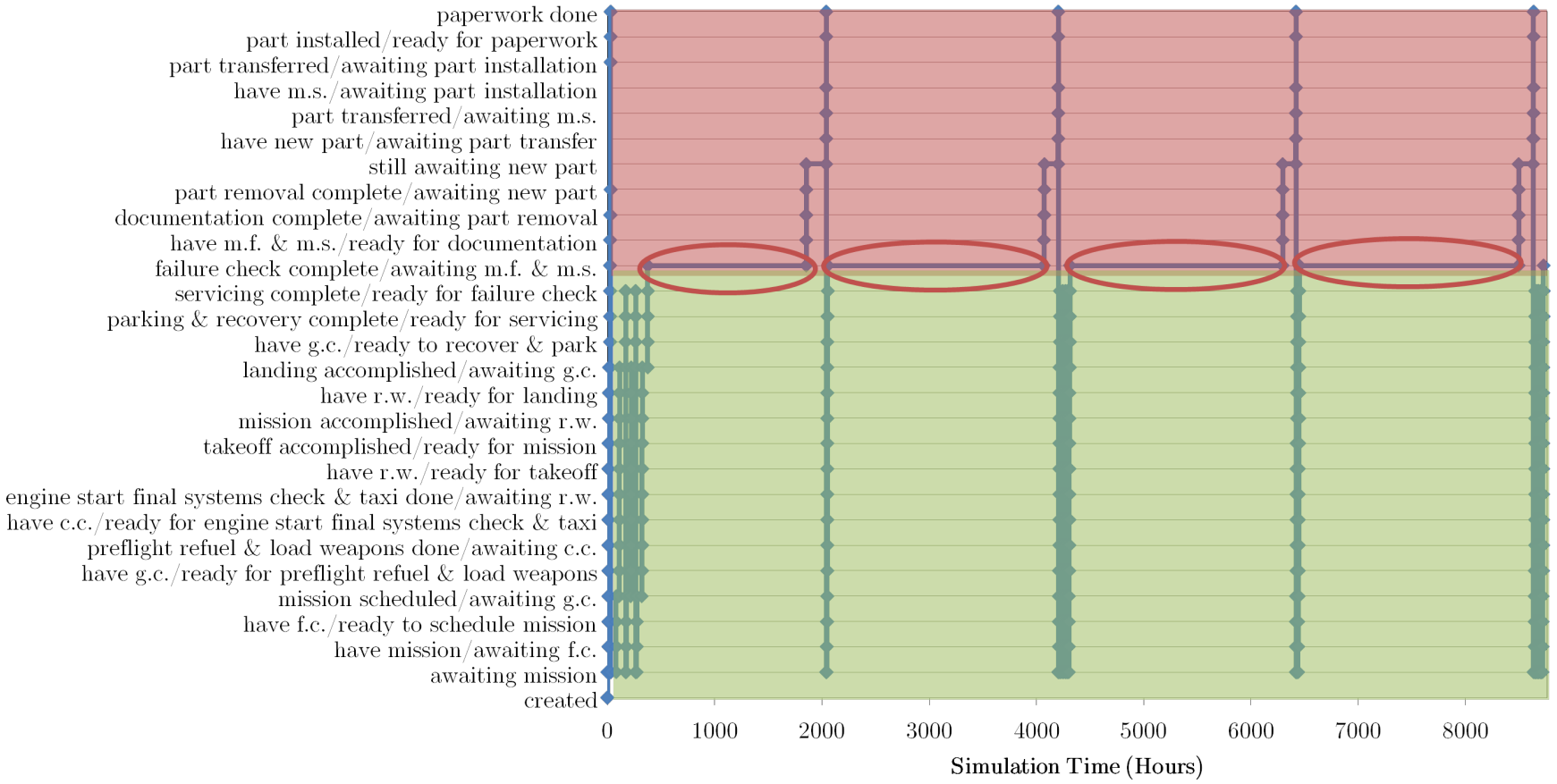


Figure 65: Event trace diagram for 25 spare parts, 365 days

4.6.4 Supply Chain Behavior

In addition to examining the aircrafts' state over time, the states of Sustain-ME's parts should also be examined at this stage to ensure that the new supply chain logic is behaving as intended. One caveat is important in introducing these plots: due to the interchangeability of parts within a given part category, Sustain-ME was constructed with each part as a unique object, assigned to a state at any given time and, when appropriate, moved from one state to another. The states are mututally exclusive and all-encompassing, so any time a part changes its state it is immediately moved from a list representing one state to another. This is helpful for checking Sustain-ME, since the length of each list can be printed as the model proceeds and the total number of parts can be tracked throughout the simulation for consistency, as well as the number within each category to verify the events being experienced by parts and aircraft. However, it requires a shift in state of mind when examining the events than any given part experiences, due to the particular way these lists were treated. Since parts are interchangeable, individual parts are not passed through the system by their ID, but rather the top part in the relevant state list is passed to the next state. Therefore the order of events that parts experience is important and must be checked against the intended part supply chain cycle, because parts move between states in a predictable pattern. However, the time at which these events occur does not necessarily reflect the timing that was experienced by a given instance of the supply chain logic, because multiple part objects might have been involved in the transfer of parts from one category to another. For instance, in completing a specific cycle of the supply chain, a part with ID number 1 might have been transferred from the state "equipped on the aircraft" to the state "shipping to depot". But while the simulation was completing its next action of pausing to emulate waiting for the part to ship, more parts might have been added to the "shipping to depot" list and would then supplant part number 1 as the top entry in this list. Then on the next step

when the same instance of the supply chain logic moved a part from the "shipping to depot" list to the "depot repair" list, the part ID number might have been 2 or 2000. Thus, when viewing the results from the perspective of an individual part, the timing might seem to make little sense.

The relative irrelevance of timing in the part state analysis means that creating plots such as those for aircraft events will not provide any helpful understanding beyond the visualization of parts continuing through the states in order. A simpler way to verify this is to perform the same check that was performed for the aircraft states. However, since parts do not experience any divergent decision points, each state can be compared against the immediately previous state to ensure that the same states follow each other each time. This is visualized by returning a value of the part number if the previous state is the expected previous state, and returning a value of 0 if it is not. Thus the logical check return value can be plotted for many parts at once and if all lines on the plot remain horizontal with no dips to the zero line, the part order matches the allowed and expected part order. Given the number of parts used in the simulation, one of these plots has been provided for each inventory level while the rest were checked and found to be fine. Another helpful way to visualize and check the part behavior is to plot the numbers of parts in each state over time.

It is also important to note that the effects of decreasing inventory are not always straightforward when viewed from the perspective of a single output metric. For instance, the total number of parts ordered or the number of repair cycles for any given part might be low for both very high and very low inventory, but high for medium inventory levels. Figures 66 through 68 demonstrate this, as they show that over the time scale the parts in the 180 spare part case go through many more repair cycles than either the 225 or 25 spare part cases. This occurs because high inventory levels have many parts are available, meaning orders are delayed or completely unnecessary for the life of the simulation. For low inventory levels few parts are available, meaning

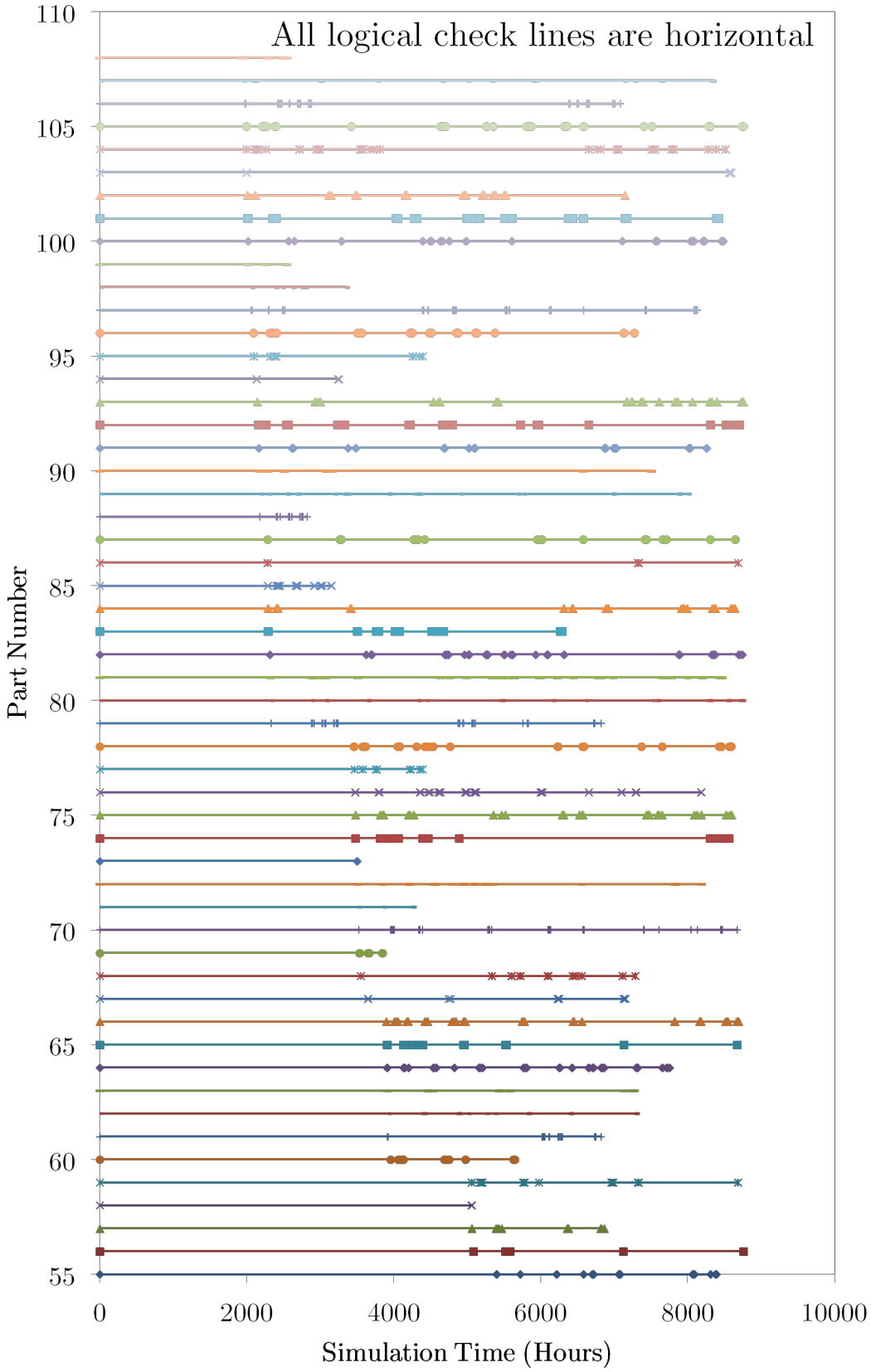


Figure 66: Part event order check with 225 spare parts

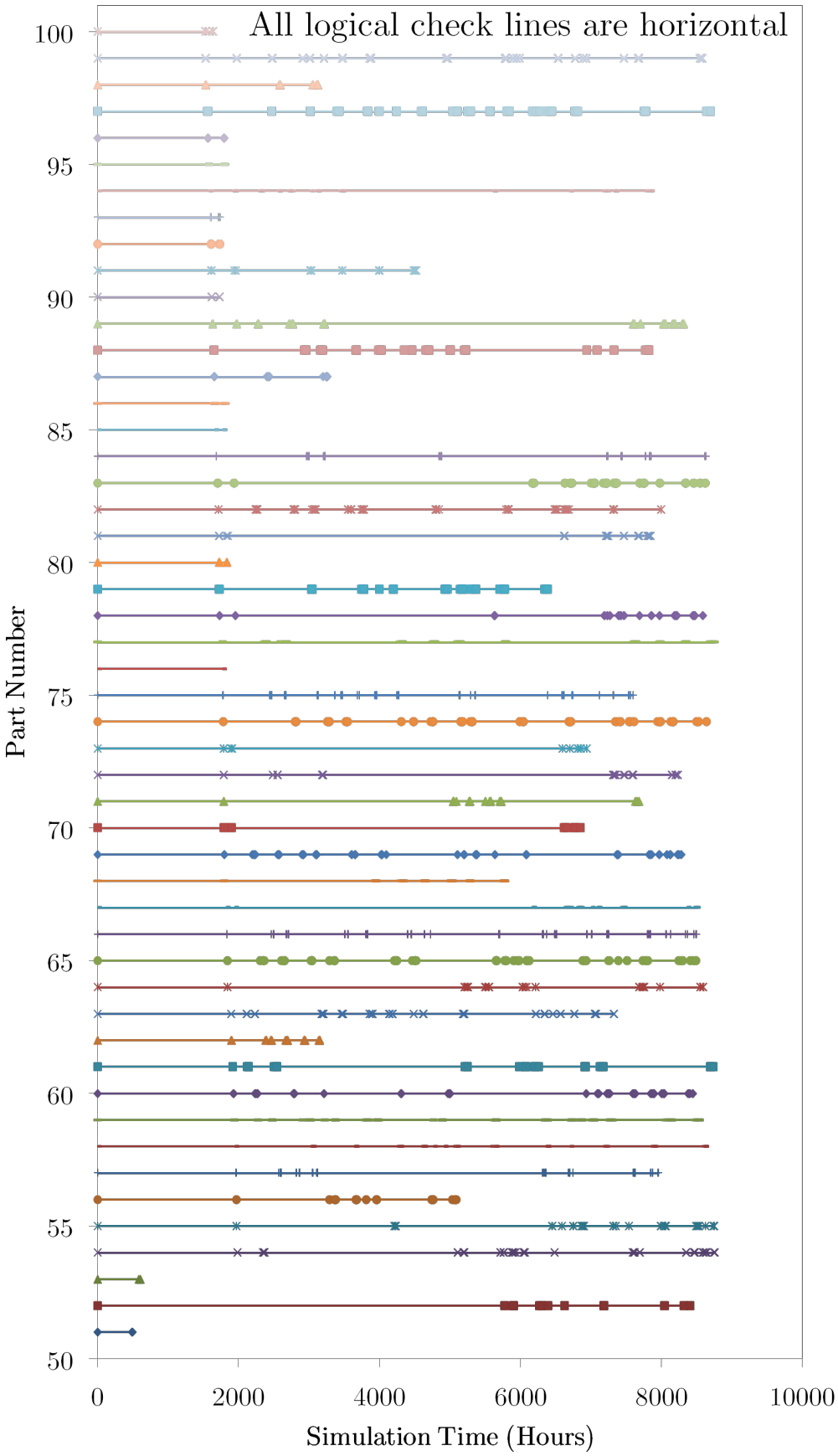


Figure 67: Part event order check with 180 spare parts

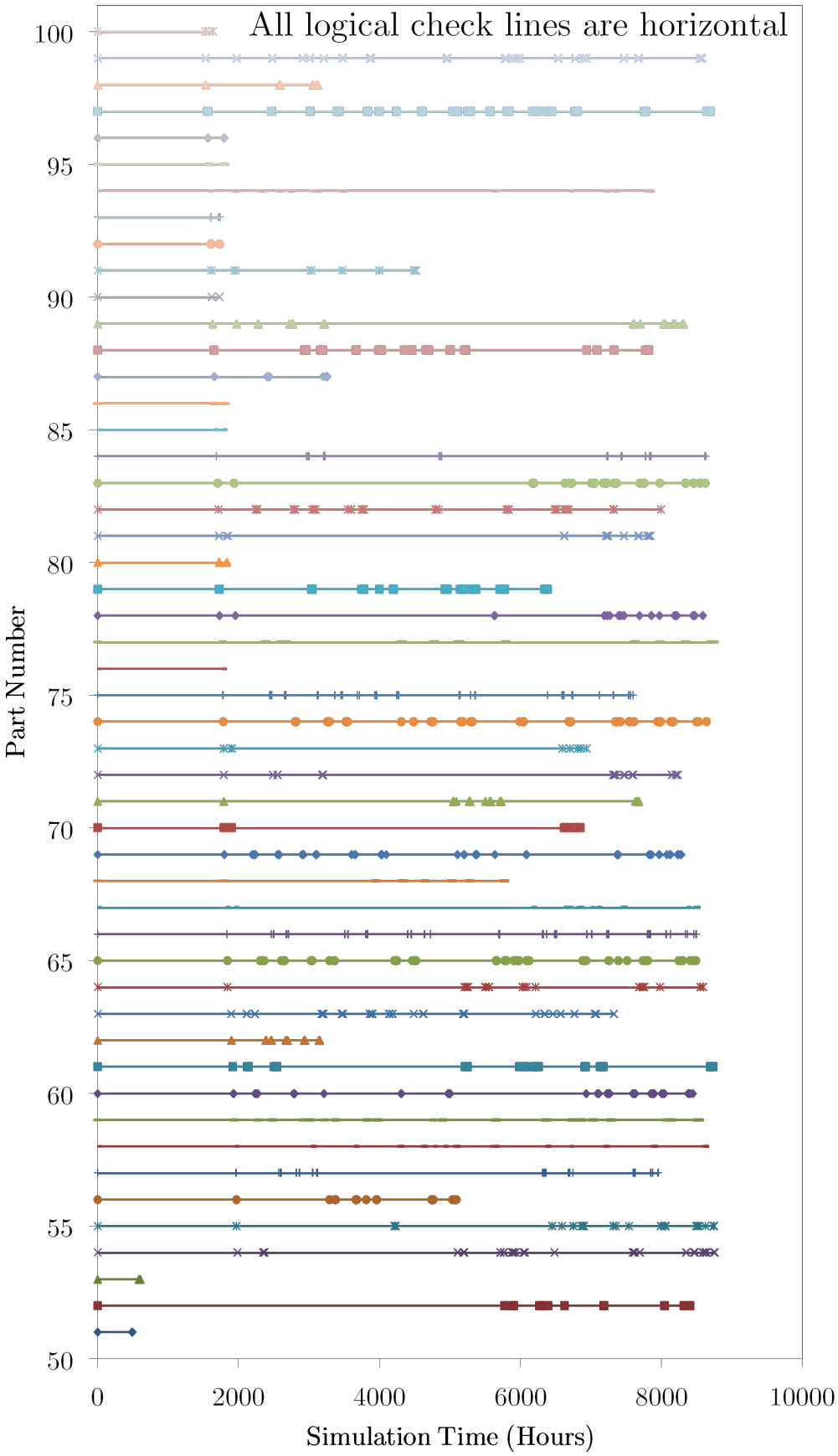


Figure 68: Part event order check with 25 spare parts

a limitation is placed on the total number of flights that can be flown until parts have been returned from depot repair; this lower level of utilization means fewer parts are ordered and fewer repair cycles completed. However, for medium levels of inventory there is just enough inventory to continue to operate much of the time, but little enough to require frequent renewals of parts. Consequently, each part goes through many more repair cycles than in either of the limiting cases.

Figures 69 through 71 demonstrate the truth of this causality. Figure 69 shows that the local inventory is never depleted, meaning that each part is used less frequently on average than for the case of Figure 70, where local inventory parts are depleted but take long enough to be used up that the depot is seeded with enough parts to provide a steady stream of them for the rest of the simulation. Figure 71 shows that this behavior is fairly different from the case of 25 spare parts, where the local inventory is depleted much more quickly, resulting in long delays between parts becoming available (as was also seen in Section 4.6.3), as evidenced by the intermittent activity in the three different state lines on the order of eighty days, the length of time required to refurbish parts. Finally, note that Figures 69 through 71 confirm that the total number of parts remains constant through the simulation, suggesting that parts are not being lost as they are passed between different states.

4.6.5 Multiple Part Categories

The final test for this portion of the code is to run Sustain-ME with multiple part categories and examine the aircraft event sequences and part event order checks as well as the numbers of parts in every state to ensure that errors are not introduced for this additional logic. Since the medium inventory case has already been discussed as the required operating conditions for the fleet, these tests will be performed at this condition. However, due to the fact that multiple part categories with different

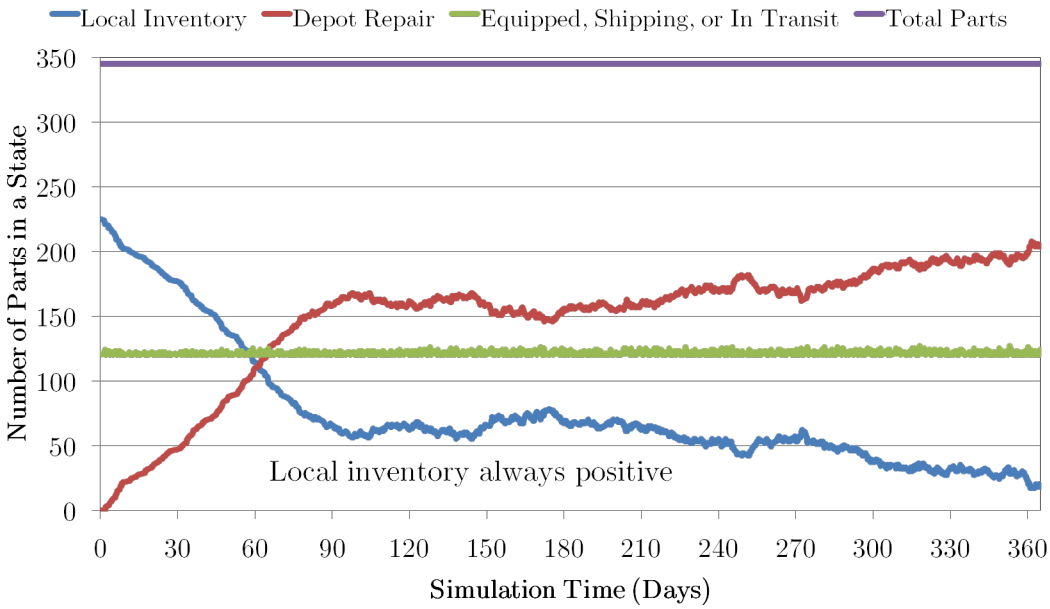


Figure 69: Number of parts in each state for 225 spare parts

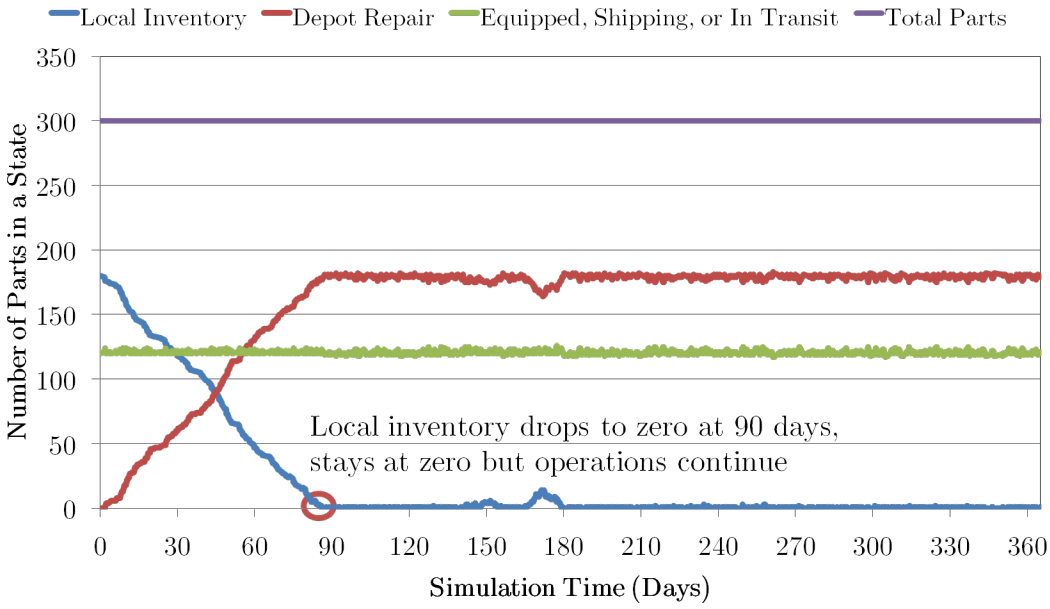


Figure 70: Number of parts in each state for 180 spare parts

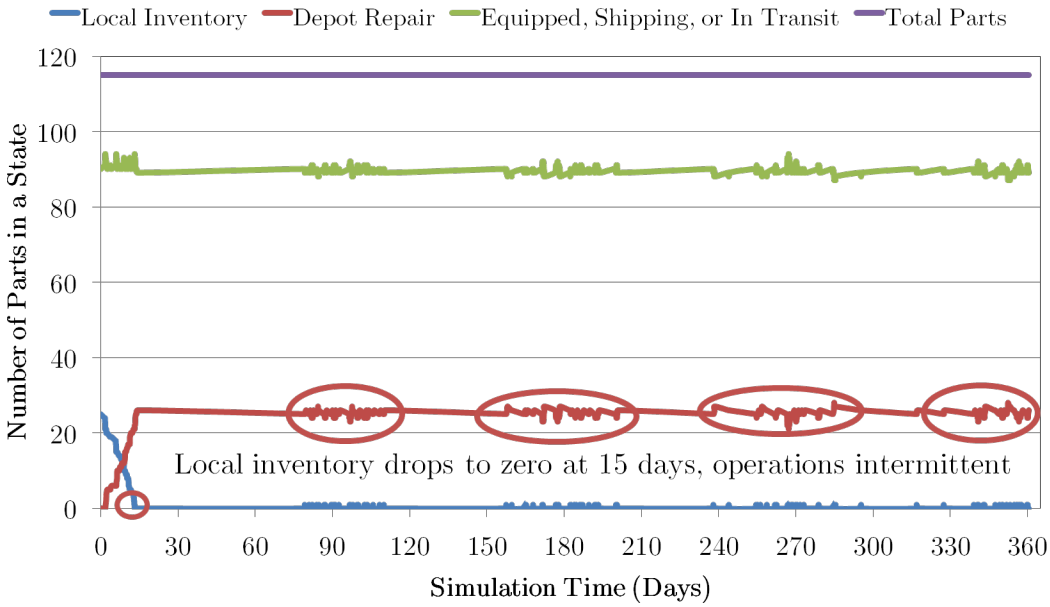


Figure 71: Number of parts in each state for 25 spare parts

reliability levels have been introduced to Sustain-ME, this medium inventory condition is different for each part. For the case of six parts with reliabilities as shown in Section 4.2, the required medium inventory is 37 for the low reliability parts and 3 for the high reliability parts.

Figure 72 shows that the features described earlier as correct fleet behavior are once again present when Sustain-ME has been expanded to six part categories. Once again the aircraft operate for the entire length of the simulation and cycle through the states in order, splitting at the known decision points. Path adherence checks reinforce that aircraft operations remain correct. Since the behavior of Sustain-ME remains in line with expected results, the next observation to be made is that part delays as were seen in Section 4.6.3 still occur. Taken together, these tests indicate that Sustain-ME behaves as expected once multiple part categories have been introduced.

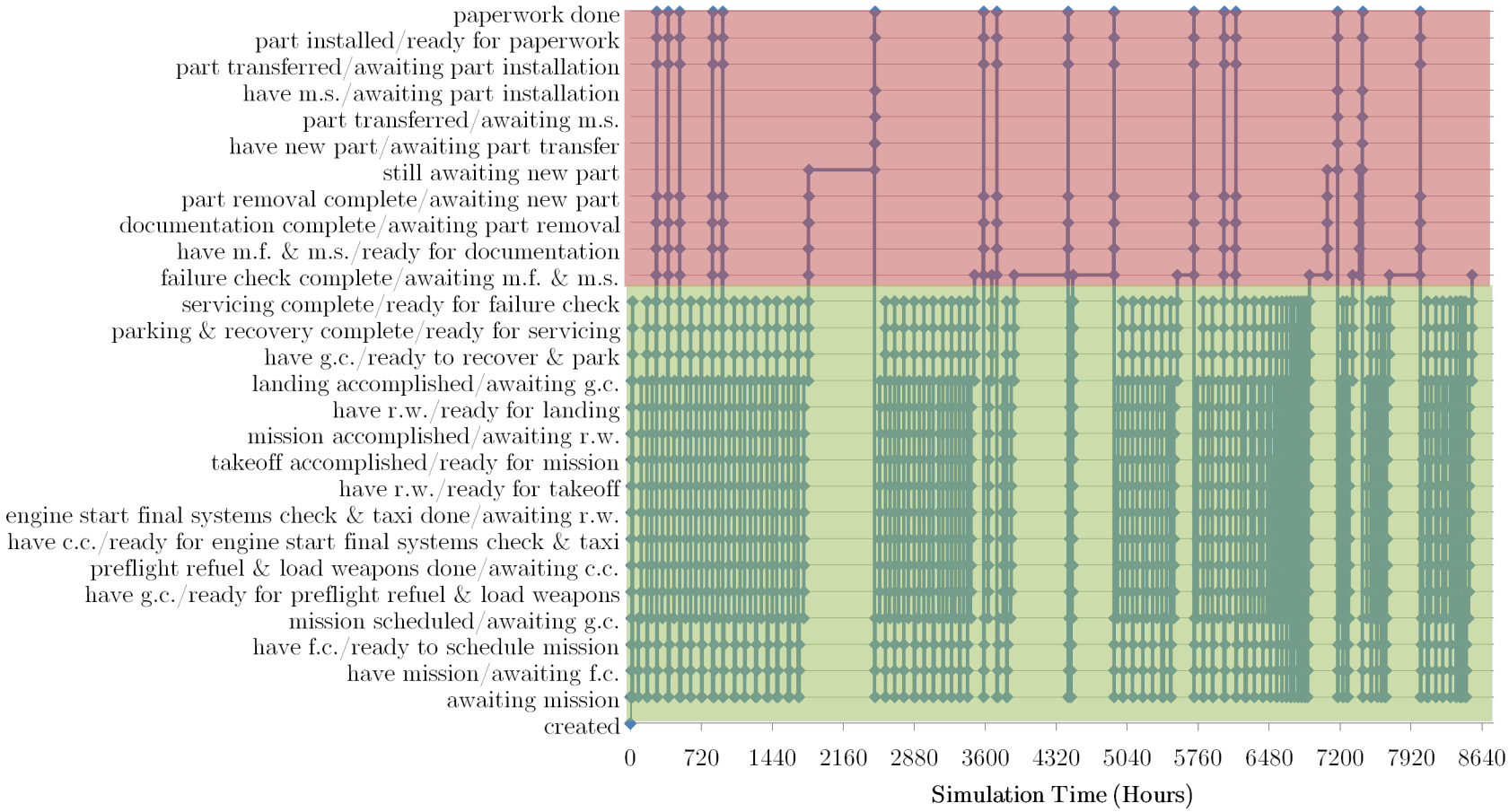


Figure 72: Event trace diagram for six parts

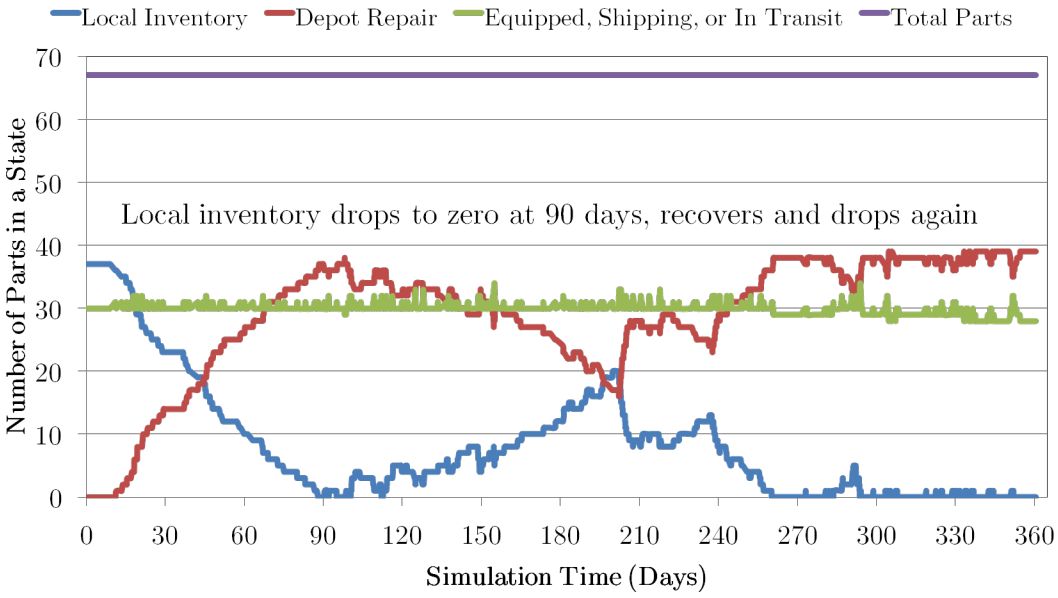


Figure 73: Number of parts in each state for low reliability part

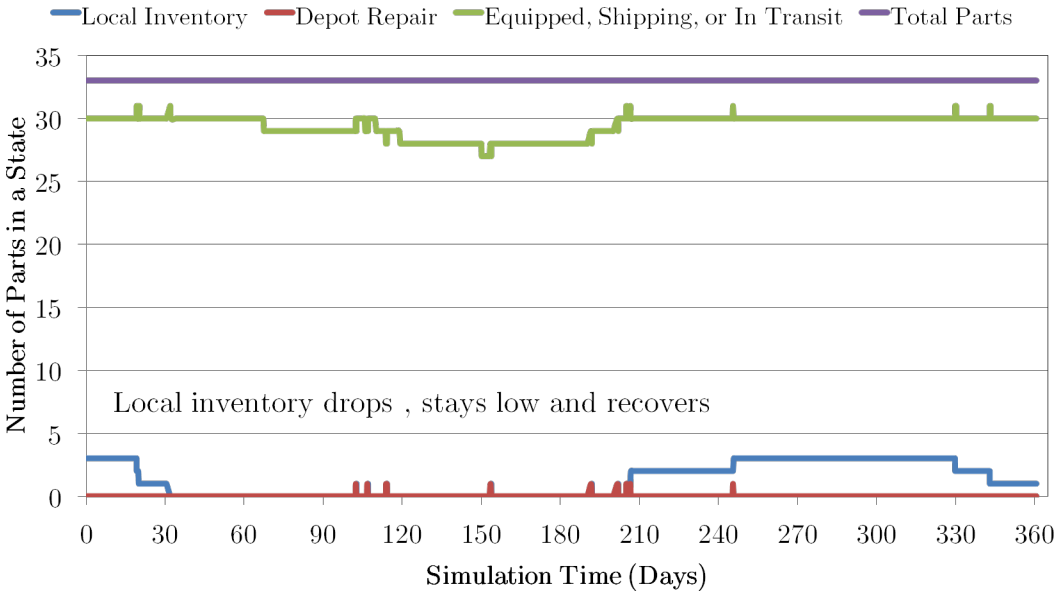


Figure 74: Number of parts in each state for high reliability part

4.6.6 Fleet Operations Including Supply Chain Conclusions

At this point an extensive amount about Sustain-ME has been tested. With the supply chain incorporated, the behavior of the aircraft and parts have been tested; the effect of changing inventory levels on the model has been observed and verified; Sustain-ME has been verified against a version of Sustain-ME with an assumed near-perfect supply chain, and the distribution of failures within the model has been checked against expected values and trends. Together, these suggest that the model with the supply chain incorporated is accurately portraying the sustainment process that has been described throughout this thesis. Furthermore, at this point the model is an accurate representation of the sustainment process under a reactive maintenance paradigm. Though more work will have to be done to incorporate the behavior of the other two maintenance paradigms, Sustain-ME at this point can be called a complete portrait of sustainment.

4.7 CBM with PHM: Verification

This section covers the modules denoted by the letter 'E' in Figure 22. The module representing a CBM maintenance paradigm as enabled by PHM was modeled based on the assumptions made in Section 3.1.4. In brief, the PHM detects that parts are failing with some amount of warning. It was assumed that the detection time is a set percentage of the part life, which is a stochastic value, and the detection time was modeled with a truncated normal distribution around the detection time which has a minimum value of zero and a maximum value of the full part life. Once detection has occurred, it was assumed that the PHM has perfect knowledge of when the part will fail.

The test for this module is to output the part life remaining and time until detection each time these values change along with the simulation time at which they change. Whenever the simulation becomes aware of a part's life (when the detection

time passes), this is output as well. Also, whenever a part fails this information is output along with whether the failure was detected in advance or not. This information was plotted for individual parts on individual aircraft. The graph should show two lines which are, generally speaking, vertically shifted from each other: the top line shows the part life remaining, and the bottom line shows the time until failure is detected. The vertical spacing between these should be the lead time in flight hours, i.e. the number of flight hours before failure occurs that failure can be detected. The horizontal spacing between when these lines come closest to the x-axis gives the lead time in real world hours. The time until detection line is confined between the part life line and the x axis, and as a result it should also remain to the left of the part life line unless the two are both being reset. If these relationships are not observed, some error has occurred in Sustain-ME. There are also three possibilities for the time relationship between events. First, the repair can either be classified as a PHM repair or as an unscheduled repair. A PHM repair means that the failure is detected before the mission on which the part fails; an unscheduled repair means that the failure is detected on the same mission as repair occurs. If the repair is unscheduled, detection, repair and installation of a new part are recorded at the same time. If the repair is a PHM repair the detection can either occur on the same mission that results in the repair, or can occur before this mission. If the events occur in any other way, this indicates that an error has occurred in Sustain-ME.

Figure 75 shows the PHM's behavior for a part with comparatively low reliability, and Figure 76 shows what this looks like for a part with comparatively high reliability.

In Figure 75, the part is replaced on the aircraft several times during the simulation. The first time this occurs, the detection occurs only a short time before the part will fail, and the part is replaced on the same mission as the failure is detected, but before the failure actually occurs. This happens again for the second replacement, when a fairly short part life is generated. On the third replacement, there is a fairly

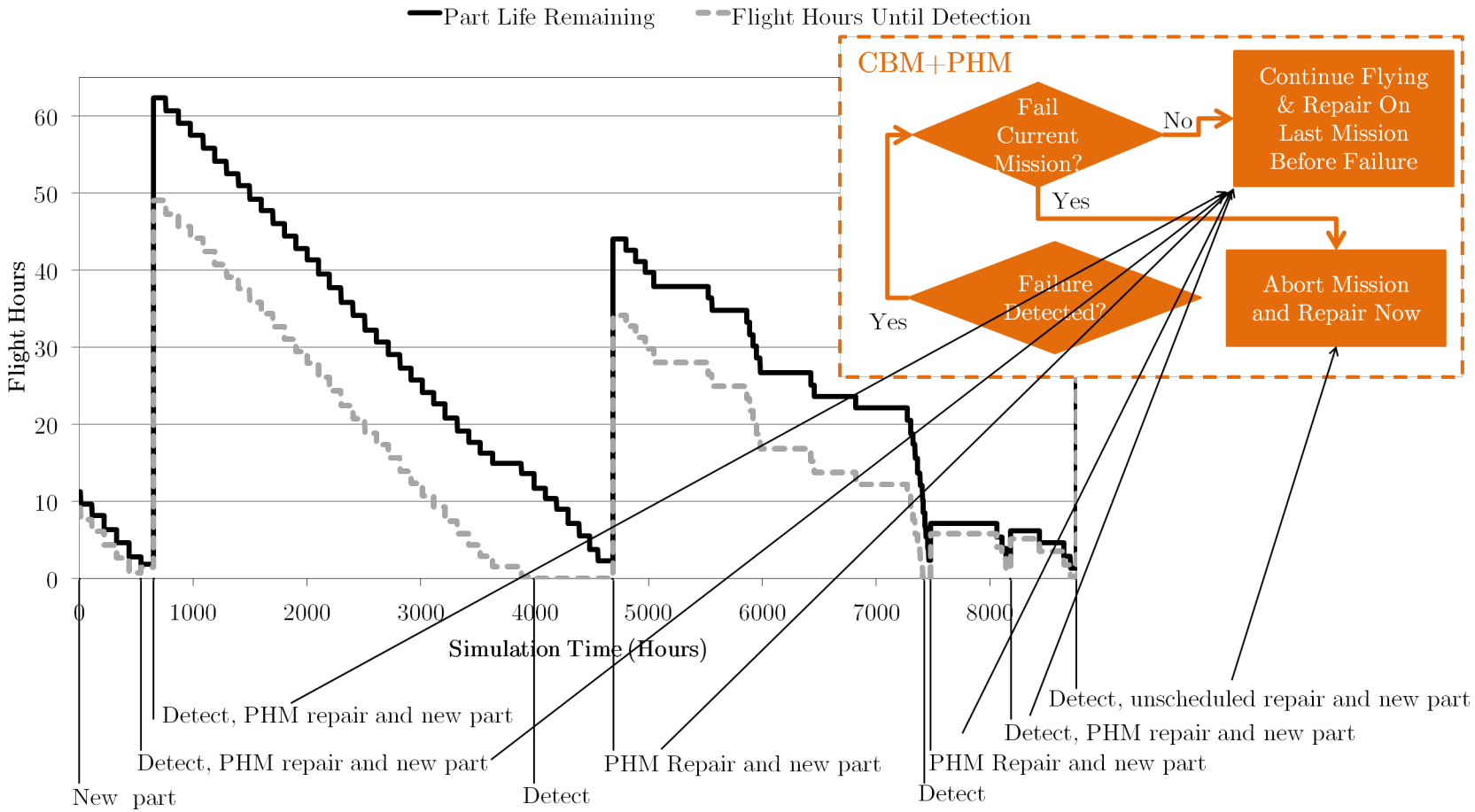


Figure 75: PHM operation for low reliability part

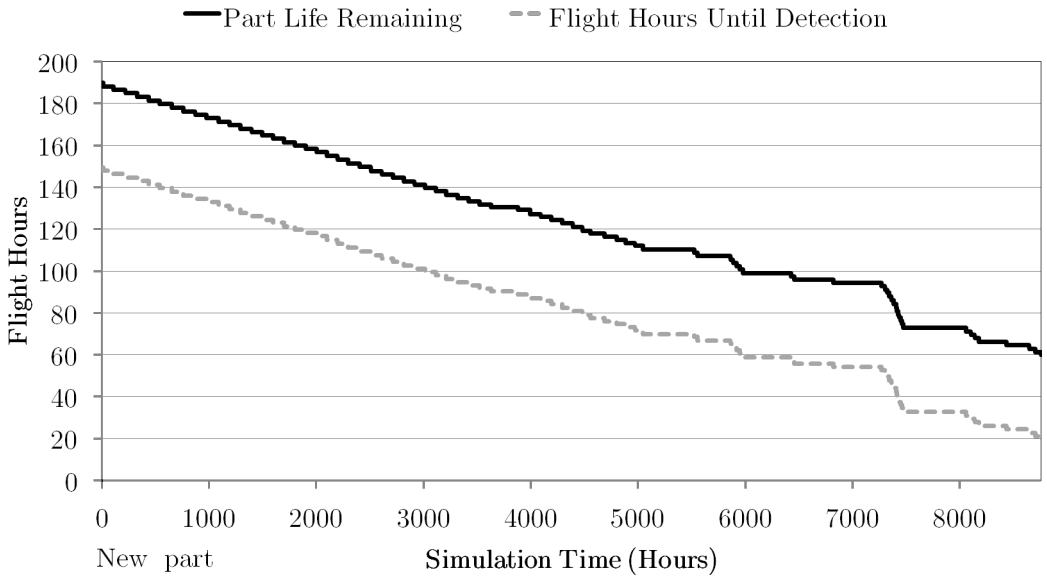


Figure 76: PHM operation for high reliability part

large lead time before failure will occur, and several missions are flown with the knowledge that the part will eventually fail. Finally, on the last replacement, the detection occurs on the same mission on which the part fails, and so an unscheduled replacement occurs. Throughout all these cycles, the predicted relationships are maintined between the part life and time remaining until detection lines, and the correct order and combination of events are observed along the timeline. In Figure 76, the part life generated is long enough that the part is never replaced. If the simulation were run longer this failure would eventually occur, and there would be a significant period over which entities within the simulation would know the part is failing but still able to be flown.

4.8 CBM-MiMOSA: Verification

This section covers the modules denoted by the letter 'F' in Figure 22. The CBM-MiMOSA module was created based on the optimization problem developed in Section 3.3.2. A preexisting open source solver, Gurobi, was used to implement this portion because it is fast and has been verified by the wider programming community. Gurobi uses an overall branch and bound technique to deal with integer variables, with the

simplex method used to evaluate each branch and bound node. As Hillier states, the combinatorial nature of mixed integer programming problems can create a huge number of solutions to solve[55]. It was therefore crucial for the simulation, which will evaluate optimal schedules frequently over the course of the simulation time, to have as effient a solver as possible. Gurobi provides this by performing several techniques to reduce the size of the problem where possible, including cutting planes, presolve, heuristics, and parallelism[50]. The output of the optimization was examined before integrating it into the full model as a means of deciding when to perform mission and maintenance scheduling. The main purpose of this examination was do verify that the solution returned by the optimizer is not only meeting the constraints as set out in the problem, but that the result of those constraints and objectives yields a reasonable solution. Thus this step was intended as much to check that the optimization problem was correctly formulated as that the optimizer returns reasonable results.

At this stage the means for using this information to replace existing mission and maintenance scheduling logic were carefully considered, as well as the determination of when to rerun the optimization to reflect changes within the simulation. Since these adaptations were not considered previously as part of Sustain-ME's development, and since the decisions made are somewhat dependent on the specifics of the fleet at any given moment, careful attention was then paid to the behavior of the model after implementing these changes. In addition to the previous information used for verification, the actions recorded and output by the PHM were updated to reflect new possibilities introduced by the optimizer.

Since the first step in verifying the optimizer was to examine the solution for a single point during the simulation, that work is reproduced here for a fleet of six aircraft rather than thirty for length considerations. Since the fleet size was reduced, the operational tempo of ten missions per day was also reduced proportionally to two missions per day. First, the optimizer inputs are shown and reorganized as the

optimizer reorganizes them. Next, these values are input into the equations outlined in Section 3.3.2 and a set of specific objective functions and constraints is generated and compared to those generated by the optimization. Finally, the solution found by the optimizer is checked for feasibility. Table 15 shows the inputs to the optimization and Table 16 shows that data reorganized.

Table 15: Unsorted optimization inputs

AC	Part 1 Life	Part 2 Life	Part 3 Life	Part 4 Life	Part 5 Life	Part 6 Life
0	100000	100000	100000	100000	100000	100000
1	100000	100000	100000	100000	44.27418	100000
2	100000	100000	0	100000	100000	100000
3	6.53144	100000	100000	100000	100000	100000
4	100000	100000	100000	12.90057	100000	100000
5	100000	100000	100000	100000	100000	100000

Recall that the modeled PHM detects the upcoming failure of one of the aircraft's parts at some point during each part's life. Once the detection occurs the PHM is assumed to have perfect knowledge of when the part will fail, so the data in Tables 15 and 16 shows whether the aircraft is available or unavailable to fly, and if available whether it is detecting failure or not. This is reflected in the value shown for each aircraft's six parts. If the PHM value is 100000, the aircraft is not detecting a failure for that part. The value 100000 acts as a stand-in for an assumed infinite part life, since this value is significantly longer than the mean time for any of the aircraft's parts and represents a theoretical maximum generated part life. If the PHM value for a part is zero, the part has failed and is currently or will soon be repaired by the simulation. If the PHM value for a part is neither zero nor 100000, the PHM detects a failure for that part, and the value given is the number of flight hours remaining before the part will fail. Recall as well that the most important part on any aircraft is the one closest to failure, since this will automatically determine its state and eventual need for repair. Thus, for aircraft with at least one part registering 0 as a part life, the aircraft is failed and any other PHM readings on the aircraft's other parts are

currently irrelevant. Similarly, for an aircraft with at least one part registering that failure is detected (but no parts actually failed), this minimum part life is the sole determiner for when maintenance will be necessary. This means that each aircraft can effectively be represented by its minimum PHM reading of part life when being considered by the optimizer. It is this logic that dictates the organization of Table 16, which is sorted by each aircraft's minimum part life, highlighted in bold in the table. However, aircraft which are down for maintenance have been moved to the end of the list so that all available aircraft appear at the beginning of the table. Thus there are three aircraft which are available and predicting failure, two aircraft which are available and not predicting failure, and one aircraft which is currently unavailable due to maintenance.

Table 16: Optimization inputs, sorted by minimum part life

AC	Part 1	Part 2	Part 3	Part 4	Part 5	Part 6	RD
3	**6.53144**	100000	100000	31.48111	100000	100000	2
4	100000	100000	100000	**12.90057**	100000	100000	3
1	100000	100000	100000	100000	**44.27418**	100000	5
0	100000	100000	100000	100000	100000	100000	N/A
5	100000	100000	100000	100000	100000	100000	N/A
2	100000	100000	**0**	100000	100000	100000	N/A

Once the number of aircraft in each of the three categories has been determined, the optimizer uses the information to set up the specific version of the optimization problem to be run. Equation 16 shows this problem for the inputs in Table 16, where the value of L_i for each aircraft in AC_{PF} is the bold value in the table. The equations are directly translated from those of Equation 15 for the specific state of the fleet defined in Tables 15 and 16, not copied from the optimization model. The variable indices are numbered not according to the aircraft ID number from the simulation, but according to the order they appear in Table 16 as this is how the optimization processes the data. The results of the optimization are then translated back into their original aircraft identification numbers for use in the simulation model.

min

$$\zeta_1 + \zeta_2 + \zeta_3 + [6.53 - 2m_{1,1} - 2m_{1,2}] + [12.9 - 2m_{2,1} - 2m_{2,2} - 2m_{2,3}]$$

$$+ [44.27 - 2m_{3,1} - 2m_{3,2} - 2m_{3,3} - 2m_{3,4} - 2m_{3,5}]$$

$$-m_{1,1} - m_{1,2} - m_{2,1} - m_{2,2} - m_{2,3} - m_{3,1} - m_{3,2} - m_{3,3} - m_{3,4} - m_{3,5}$$

$$-m_{4,1} - m_{4,2} - m_{4,3} - m_{4,4} - m_{4,5} - m_{5,1} - m_{5,2} - m_{5,3} - m_{5,4} - m_{5,5}$$

subject to

$$6.53 - 2m_{1,1} - 2m_{1,2} \geq 0$$

$$12.9 - 2m_{2,1} - 2m_{2,2} - 2m_{2,3} \geq 0$$

$$44.27 - 2m_{3,1} - 2m_{3,2} - 2m_{3,3} - 2m_{3,4} - 2m_{3,5} \geq 0$$

$$2 - m_{1,1} - m_{2,1} - m_{3,1} - m_{4,1} - m_{5,1} \geq 0$$

$$2 - m_{1,2} - m_{2,2} - m_{3,2} - m_{4,2} - m_{5,2} \geq 0$$

$$2 - m_{2,3} - m_{3,3} - m_{4,3} - m_{5,3} \geq 0$$

$$2 - m_{3,4} - m_{4,4} - m_{5,4} \geq 0$$

$$2 - m_{3,5} - m_{4,5} - m_{5,5} \geq 0 \tag{16}$$

$$\zeta_1 - (T_1 - 36) \geq 0$$

$$\zeta_1 + (T_1 - 36) \geq 0$$

$$\zeta_2 - (T_2 - T_1 - 36) \geq 0$$

$$\zeta_2 + (T_2 - T_1 - 36) \geq 0$$

$$\zeta_3 - (T_3 - T_2 - 36) \geq 0$$

$$\zeta_3 + (T_3 - T_2 - 36) \geq 0$$

$$0 \leq m_{1,1} \leq 1\,,\ 0 \leq m_{1,2} \leq 1\,,\ 0 \leq m_{2,1} \leq 1\,,\ 0 \leq m_{2,2} \leq 1\,,\ 0 \leq m_{2,3} \leq 1$$

$$0 \leq m_{3,1} \leq 1\,,\ 0 \leq m_{3,2} \leq 1\,,\ 0 \leq m_{3,3} \leq 1\,,\ 0 \leq m_{3,4} \leq 1\,,\ 0 \leq m_{3,5} \leq 1$$

$$0 \leq m_{4,1} \leq 1\,,\ 0 \leq m_{4,2} \leq 1\,,\ 0 \leq m_{4,3} \leq 1\,,\ 0 \leq m_{4,4} \leq 1\,,\ 0 \leq m_{4,5} \leq 1$$

$$0 \leq m_{5,1} \leq 1\,,\ 0 \leq m_{5,2} \leq 1\,,\ 0 \leq m_{5,3} \leq 1\,,\ 0 \leq m_{5,4} \leq 1\,,\ 0 \leq m_{5,5} \leq 1$$

$$39.8 \leq T_1 \leq 63.8\,,\ 63.8 \leq T_2 \leq 87.8$$

$$111.8 \leq T_3 \leq 135.8$$

$m_{i,j}$ integer $\forall i \leq 5, j \leq min(MD_i, 5)$

```
Minimize
  obj1 + obj2 - obj3
Subject To
 cobj1: - zeta_0 - zeta_1 - zeta_2 + obj1 = 0
 cobj2: 2 m_0_0 + 2 m_0_1 + 2 m_1_0 + 2 m_1_1 + 2 m_1_2 + 2 m_2_0 + 2 m_2_1
   + 2 m_2_2 + 2 m_2_3 + 2 m_2_4 + obj2 = 63.70618709901537
 cobj3: - m_0_0 - m_0_1 - m_1_0 - m_1_1 - m_1_2 - m_2_0 - m_2_1 - m_2_2
   - m_2_3 - m_2_4 - m_3_0 - m_3_1 - m_3_2 - m_3_3 - m_3_4 - m_4_0 - m_4_1
   - m_4_2 - m_4_3 - m_4_4 + obj3 = 0
 c0_0: m_0_0 + m_1_0 + m_2_0 + m_3_0 + m_4_0 <= 2
 c0_1: m_0_1 + m_1_1 + m_2_1 + m_3_1 + m_4_1 <= 2
 c0_2: m_1_2 + m_2_2 + m_3_2 + m_4_2 <= 2
 c0_3: m_2_3 + m_3_3 + m_4_3 <= 2
 c0_4: m_2_4 + m_3_4 + m_4_4 <= 2
 c1_0: 2 m_0_0 + 2 m_0_1 <= 6.53143973961157
 c1_1: 2 m_1_0 + 2 m_1_1 + 2 m_1_2 <= 12.9005650077877
 c1_2: 2 m_2_0 + 2 m_2_1 + 2 m_2_2 + 2 m_2_3 + 2 m_2_4 <= 44.2741823516161
 c2.1_0: - T_0 + zeta_0 >= -36
 c2.2_0: T_0 + zeta_0 >= 36
 c2.1_1: T_0 - T_1 + zeta_1 >= -36
 c2.2_1: - T_0 + T_1 + zeta_1 >= 36
 c2.1_2: T_1 - T_2 + zeta_2 >= -36
 c2.2_2: - T_1 + T_2 + zeta_2 >= 36
Bounds
 39.8 <= T_0 <= 63.8
 63.8 <= T_1 <= 87.8
 111.8 <= T_2 <= 135.8
Binaries
 m_0_0 m_0_1 m_1_0 m_1_1 m_1_2 m_2_0 m_2_1 m_2_2 m_2_3 m_2_4 m_3_0 m_3_1
 m_3_2 m_3_3 m_3_4 m_4_0 m_4_1 m_4_2 m_4_3 m_4_4
End
```

Figure 77: Optimization module problem statement

Figure 77 shows SutstainME as defined by the optimization module when the data defined in Table 15 are provided as inputs. One difference is notable: the optimization module starts its indexing with 0 rather than 1, so all the indices are one less than those shown in Equation 16. However, aside from this cosmetic difference, the variables required to define the problem, the specific constants used in objective functions and constraints, and the bounds of the continuous design variables match those shown in Equation 16. This indicates that, as a first check, the optimization module is correctly coding the optimization problem.

The optimization module's solution to the problem is shown in Tables 17, 18 and 19 along with the values of constraints associated with those decision variables. For the missions, constraints are mostly associated with the summation of the missions in different dimensions, and the structures of Tables 17 and 18 reflect this. The

constraints associated with summing all missions flown on a given day are presented along the bottom of Table reft:mOut1, and the value of this summation is presented in the row above, below the values being summed. The constraints associated with summing all missions flown by a given aircraft are presented along the bottom of Table 18, and the value of this summation is presented in the row above, beneath the values being summed. One further constraint, that the mission variables are restricted to binary settings, can be confirmed visually in the middle portion of both tables. Constraints which are highlighted green are constraints which are met by the problem; in this case all constraints are satisfied for both sets of decision variables.

Table 17: Mission settings for optimized problem

	Day 1	Day 2	Day 3	Day 4	Day 5
AC 3	0	1			
AC 4	1	1	1		
AC 1	1	0	1	1	1
AC 0	0	0	0	1	1
AC 5	0	0	0	0	0
$\sum_{i'=1}^{AC_{Avail}} m_{i.j}$	2	2	2	2	2
$\leq OT_j$	$OT_1 = 2$	$OT_2 = 2$	$OT_3 = 2$	$OT_4 = 2$	$OT_5 = 2$

Table 18: Mission settings for optimized problem

	AC 3	AC 4	AC 1	AC 0	AC 5
Day 1	0	1	1	0	0
Day 2	1	1	0	0	0
Day 3		1	1	0	0
Day 4			1	1	0
Day 5			1	1	0
$\sum_{j=1}^{MD_i}(m_{i,j}d)$	2	6	8	4	0
$\leq L_i$	$L_3 = 6.5$	$L_4 = 12.9$	$L_1 = 44.3$		

Table 19: Maintenance time settings for optimized problem

Maintenance Time	Lower Bound	Optimum Value	Upper Bound
T_1	$39.8 \geq$	39.8	≤ 63.8
T_2	$63.8 \geq$	75.8	≤ 87.8
T_3	$111.8 \geq$	111.8	≤ 135.8

Having shown that the optimizer has returned a feasible solution, the objective function values at the optimum will be discussed. The maintenance times selected make a great deal of sense, since they are spaced from each other by exactly the mean time between failures, and are as far apart from one another as the constraints allow. At these values, the objective function associated with the spacing of maintenance events is zero, its minimum possible value. For these maintenance times, the part life wasted for each part is about 4.5, 7, and 36 hours for the three parts for a total of about 48 hours wasted. This same value could have been achieved if aircraft 3 had been flown on the first day of the optimization instead of aircraft 4 since the two additional hours taken from aircraft 3's part would have been added onto aircraft 4's part; this suggests that will be several equally good solutions to the problem in some situations. Future efforts might take advantage of this fact by adding additional information to the optimization about the other parts equipped on the aircraft, trying to wear them out faster or slower depending on what is desirable to the overall behavior of the fleet. However, such fine-tuning is beyond the scope of this particular problem. Finally, the total number of missions flown reached its maximum possible value with the full operational tempo flown over this five day period. However, keep in mind that this solution was returned when five of the six aircraft were available to fly missions; at other conditions this might not be possible.

The next step in verifying the optimizer's behavior within the simulation was to examine the behavior of several different entities within Sustain-ME with the optimizer in place. As was done for the PHM module, the simulation's decision making process for performing repairs is reviewed. The data provided to the optimization at each point in time was also reviewed to verify that it matches the knowledge available to the PHM at any given moment; this was found to be the case but difficult to visualize because of the relatively small magnitude of the part lives at detection compared to the 100000 value used as the default for parts not detecting failure. Consequently

only the part life remaining and flight hours until detection were plotted in Figure 78, along with the major events which accompanied any strong transitions.

Figure 78 shows that several obvious behavior violations are not occurring: replacements are not occurring before detection; unschedule maintenance is occurring immediately after detection, not after detection and then flying several missions; and detection time is never greater than part life or less than zero. Beyond these obvious signs of correct behavior, the plot shows that in some cases, replacement is occurring sooner than it would have under a pure PHM model, where parts are only replaced when they will fail on the next mission. However, at least for the part and aircraft shown, the part life wasted as represented by the distance below a line that moves vertically upward on the graph is not egregiously large. This indicates that the simulation is respecting the desire to fly missions and use part life as well as to evenly space maintenance visits.

Figure 79 shows the behavior for a different part with high reliability. Due to the part's much higher probability of generating long part lives, the cycle time to repair parts is much lower and this particular part was in use for almost all of the simulation before requiring replacement. However, once failure was detected for this part it was immediately replaced by the optimizer, even though there were about 25 flight hours remaining that the part could have flown. This occurred because the optimizer was told to fix the aircraft during a period fairly soon after detection as per the assumptions of the optimization problem. This highlights one of the sacrifices made to implement maintenance optimization: some parts will be repaired long before they need to in order to achieve more even spacing of maintenance visits. However, this is still preferable to a scheduled maintenance paradigm where parts are replaced even if no indication has been obtained that parts may soon fail; in this case the part is at least partially degraded if it is registering an upcoming failure.

The final step in verifying the inclusion of the optimizer in the modeling logic is to

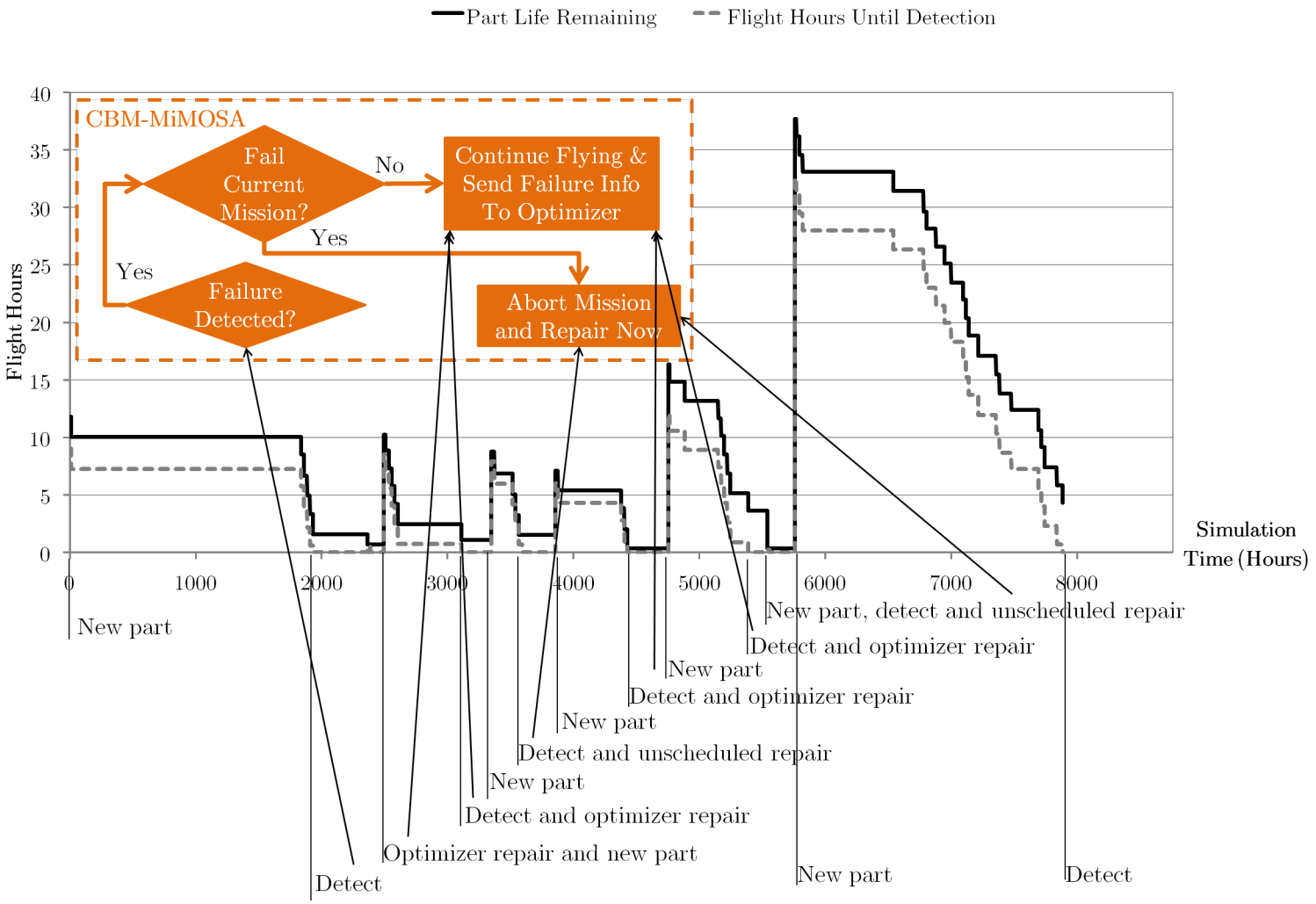


Figure 78: PHM operation under optimization for low reliability part

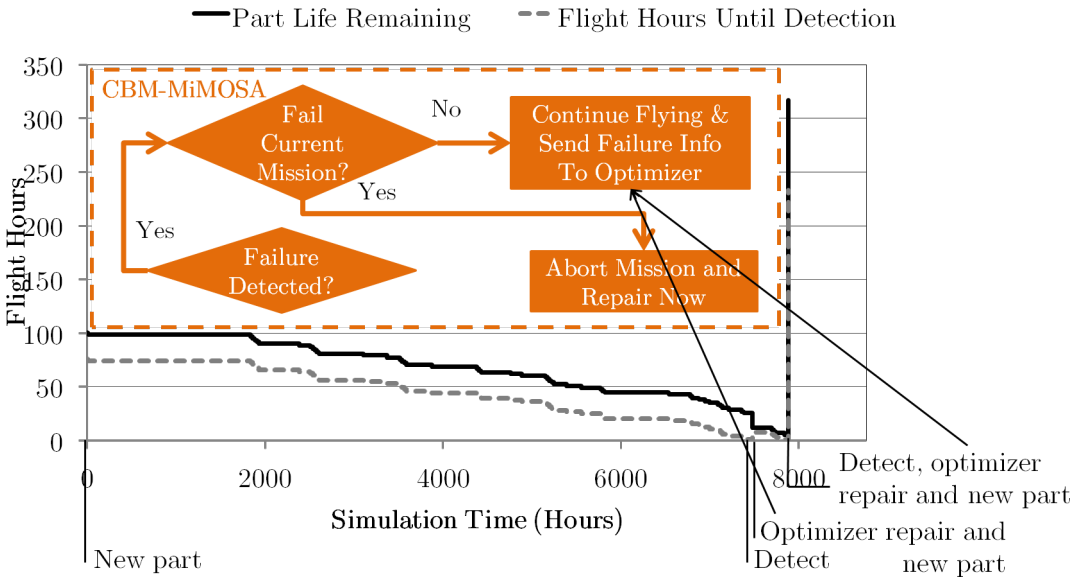


Figure 79: PHM operation under optimization for high reliability part

re-examine the states of the fleet's aircraft over time; however, before that can happen a discussion of how the aircraft operating logic changed should first occur. For the most part the aircraft's operating logic remained unchanged, since the PHM module works within the aircraft's code but is largely separate from its operations; it affects the decision to enter certain portions of the operational logic, but not what those steps actually are. However, one necessary change needed to be introduced because of the way the optimizer is able to schedule aircraft to missions. The optimizer has the freedom to formulate its schedule so that aircraft do not fly on the day they will be repaired. Under previous logic, aircraft are repaired immediately upon experiencing any sort of failure, but with the possibility to delay a maintenance visit by a few days to increase steadiness of maintenance visits a new logical check had to be created. This decision point determines whether an aircraft which is in the "awaiting mission" state has a maintenance event scheduled during the current day. If this is so, the aircraft is reactivated for the purpose of completing the maintenance loop only, so it skips all the states associated with flying a mission and enters the maintenance cycle, after which it once again returns to an "awaiting mission" state. Since this decision

point leads into another decision point based on whether parts are available, two new loops are created by adding this one decision point. The aircraft operational logic is shown in Figure 80 with the updated logic.

The loops shown in Figures 33 through 35 now look like the loops represented in Figures 84 through 83.

Given the fact that the aircraft operational order of events has been largely confirmed at this point, the check for aircraft behavior will focus instead on comparing the steps encountered by the aircraft in the simulation against the known paths. Figure 86 shows this, where the amount of time spent in any given path can be determined by its length on the time axis. If the aircraft fails to adhere to one of the known behavioral loops, a gap will appear on the plot to indicate a failure which must be investigated. Other potential sources of concern could arise if aircraft spend too long completing a loop, and if this is the case the specific behavior of that loop can be delved into further.

In Figure 86, the line is unbroken meaning that the aircraft does not deviate from its alotted choice of paths and states. Note, however, that not all paths represented by specific values on the y axis are visit in this plot. In some cases, especially the paths that are less likely to be traveled, the specific conditions that trigger a path may not occur for a given aircraft during the simulation time. If this path never appears for any of the aircraft this could potentially be cause for concern, but in this case path 5 was visited by other aircraft during the simulation. Thus the aircraft behavior is confirmed to match what was predicted under optimization. At this point, a reasonable degree of confidence in the modeling of CBM-MiMOSA can be upheld.

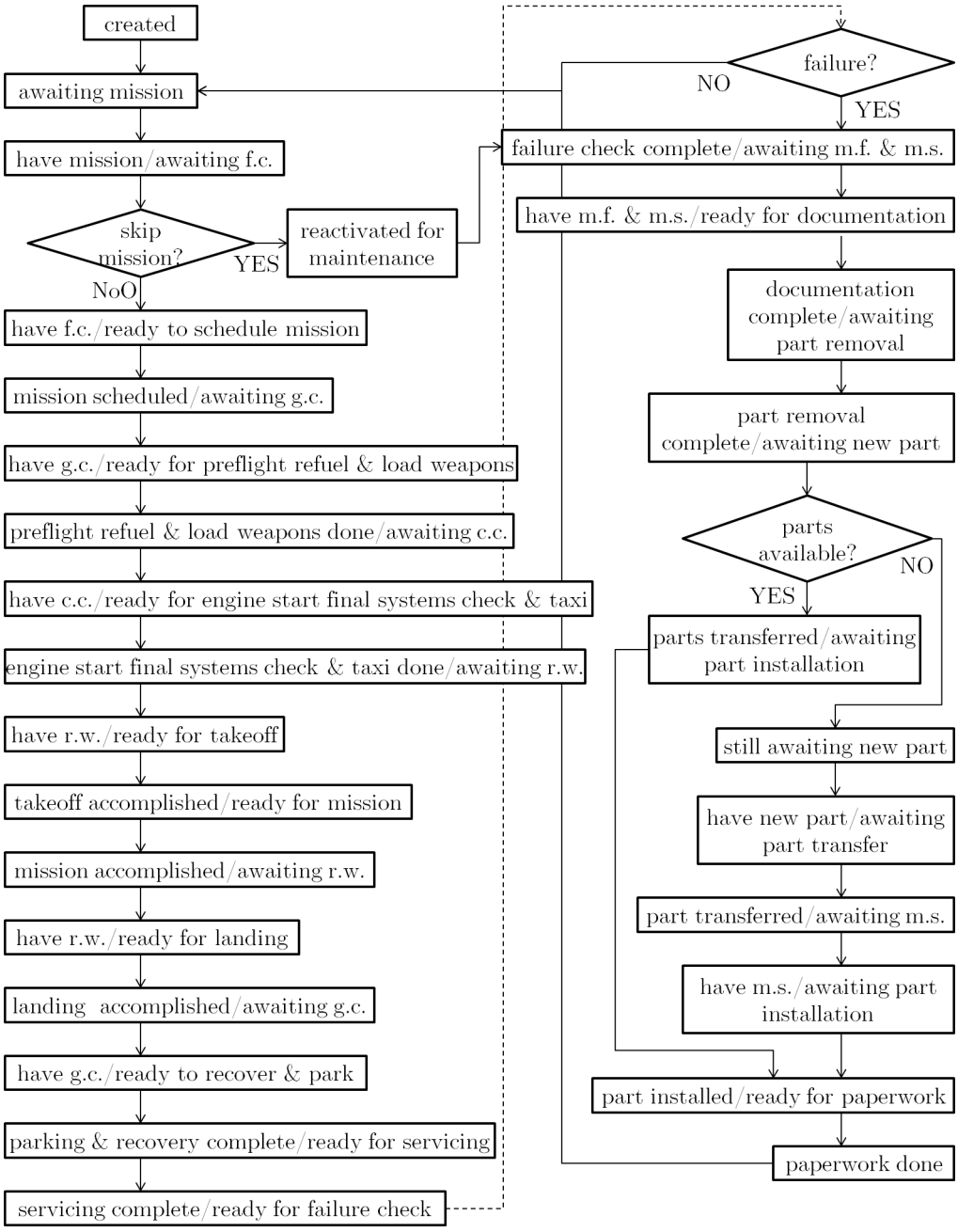


Figure 80: Updated aircraft operational logic

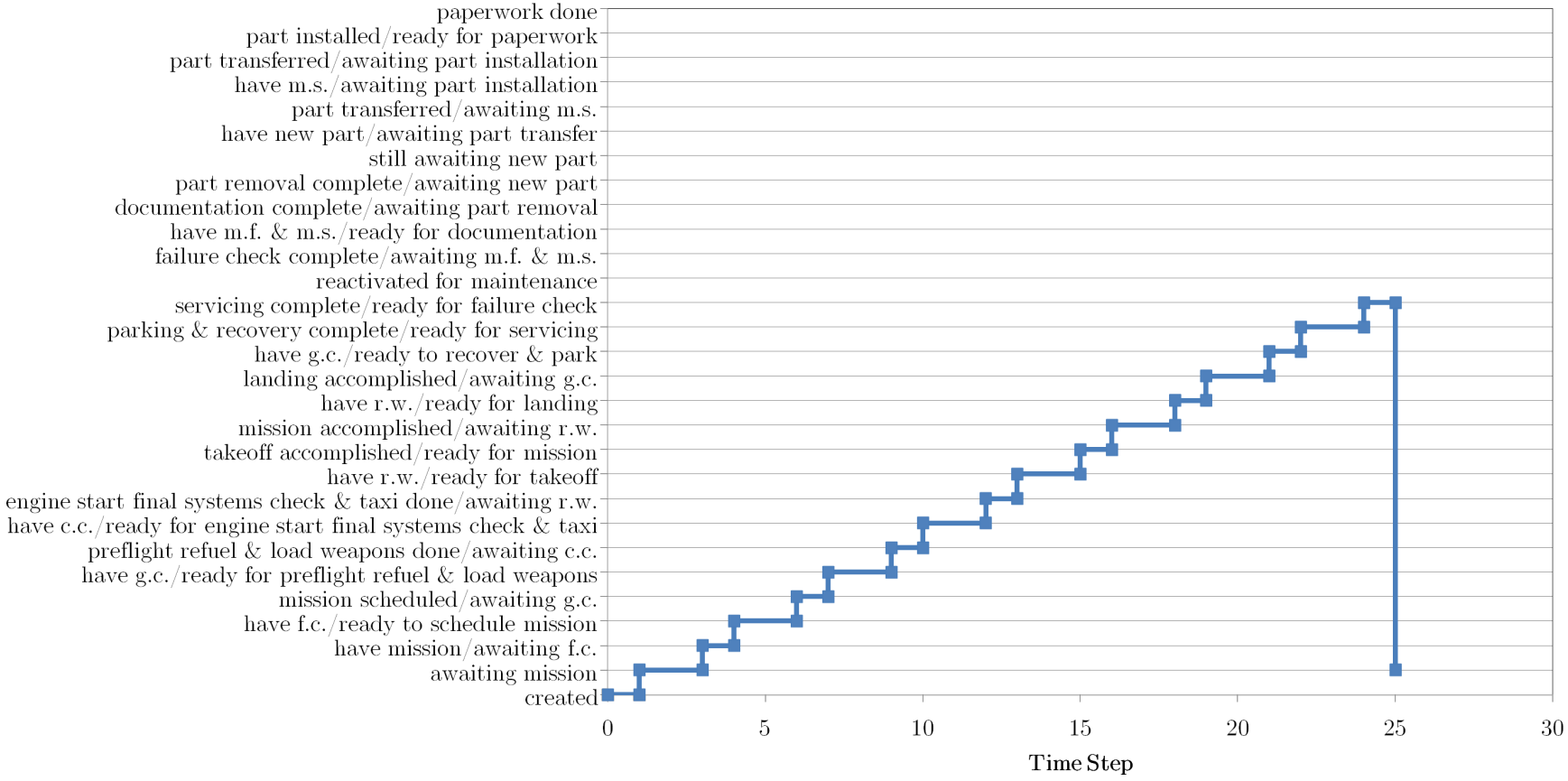


Figure 81: Path 1 aircraft order of events

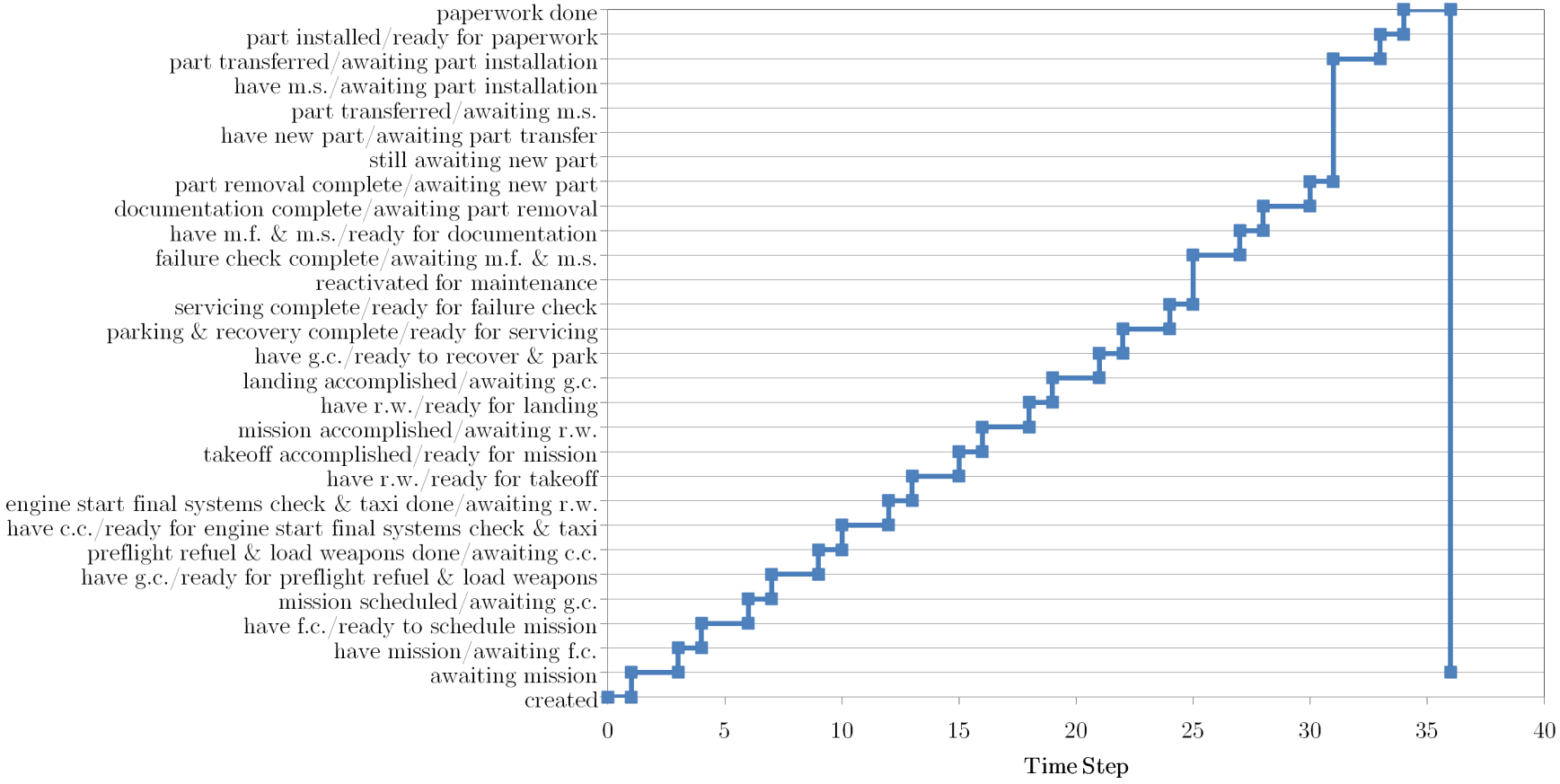


Figure 82: Path 2 aircraft order of events

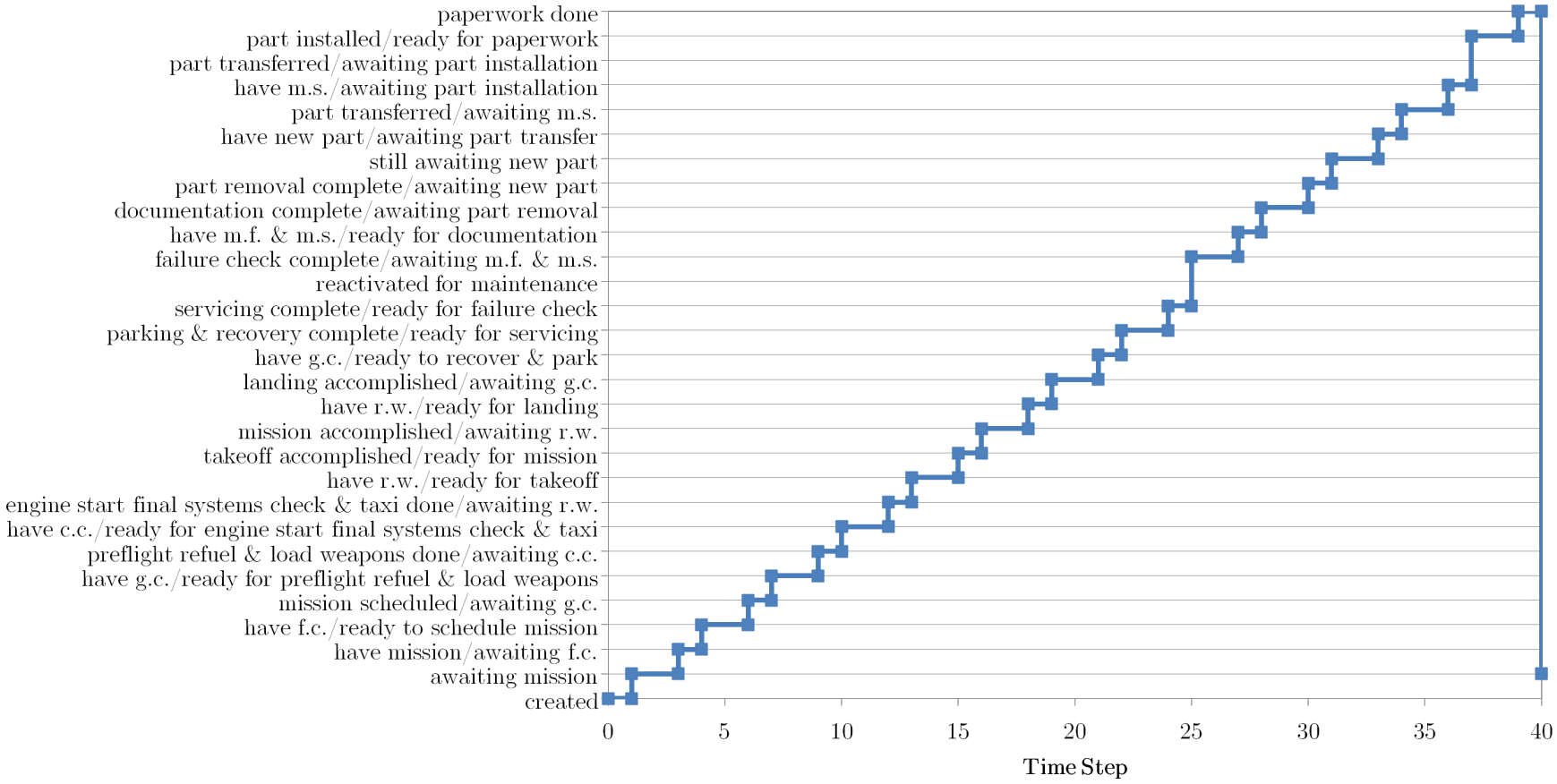


Figure 83: Path 3 aircraft order of events

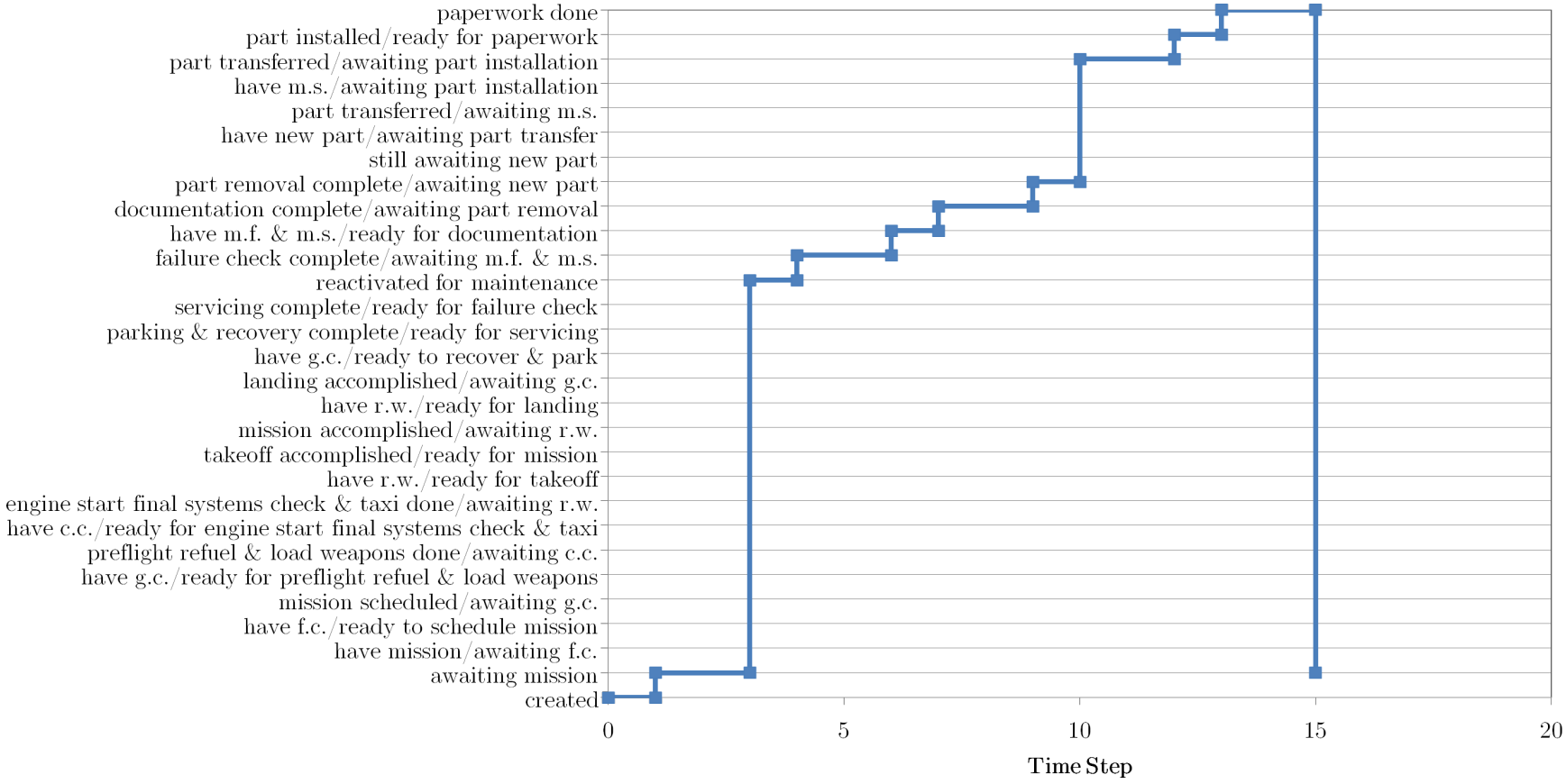


Figure 84: New path 4 aircraft order of events

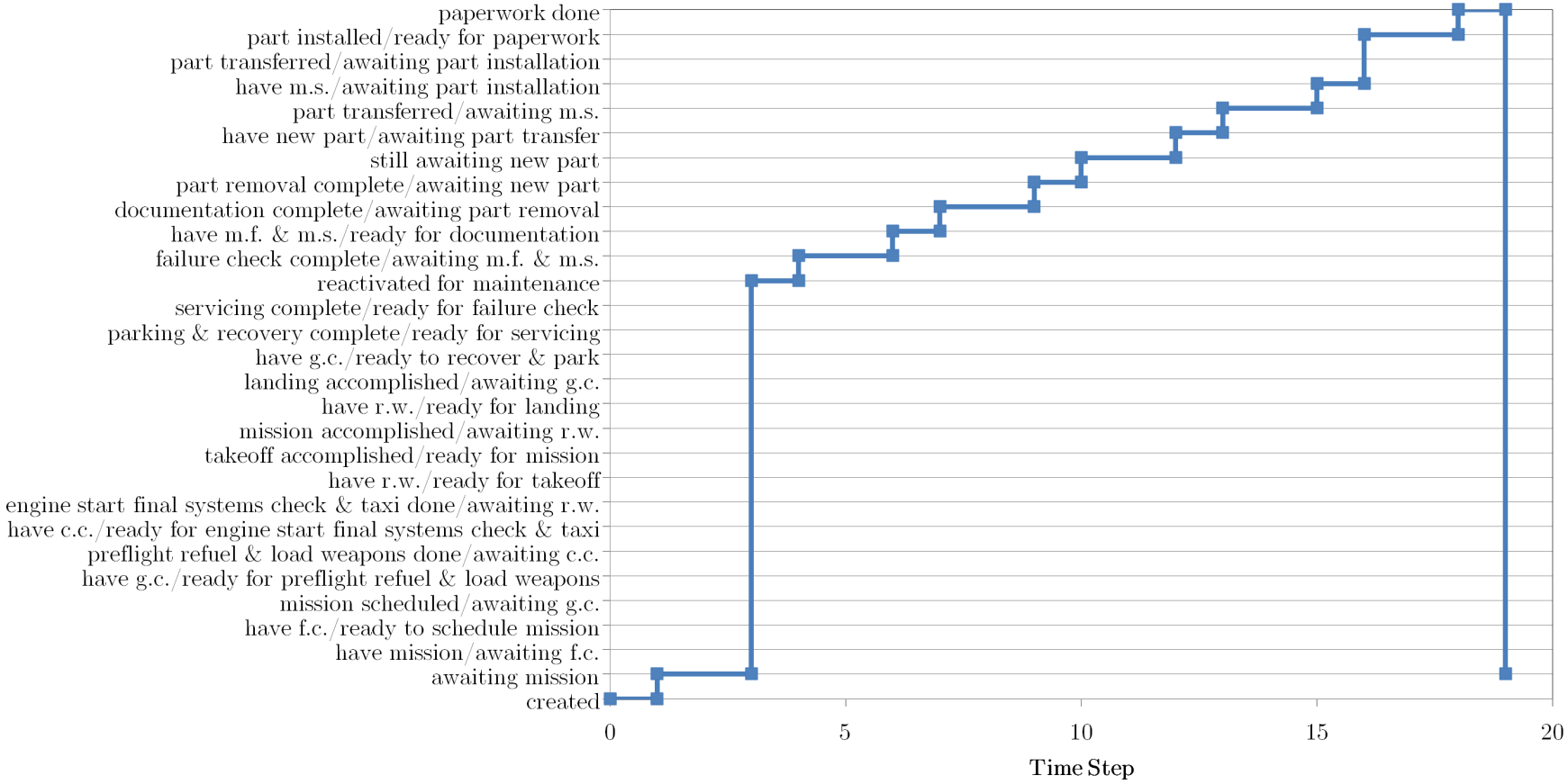


Figure 85: New path 5 aircraft order of events

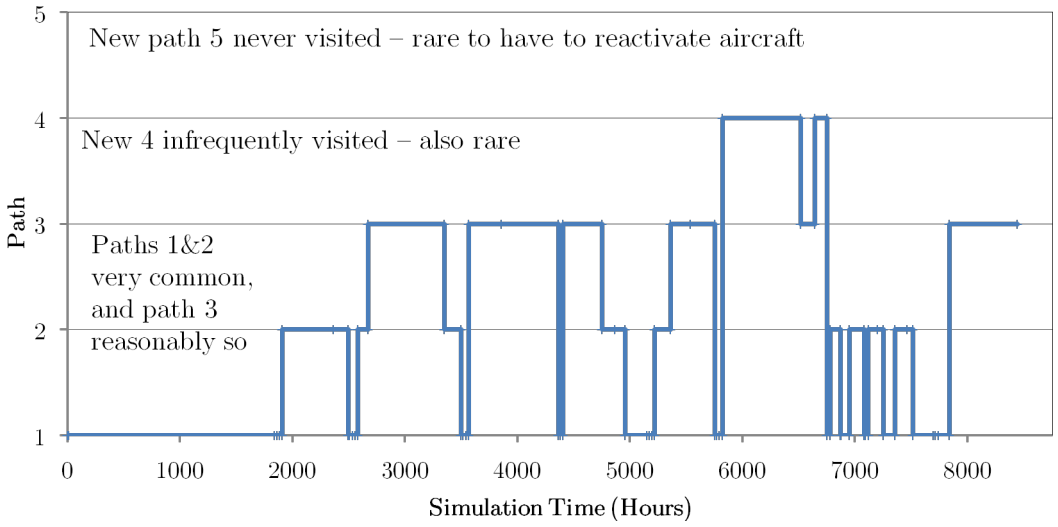


Figure 86: Order of paths for one aircraft under optimization

4.9 Verification Conclusions

With the inclusion of the optimization in Sustain-ME, all of the maintenance paradigms have now been incorporated into Sustain-ME, and their behvior verified. Through use of many different tests, the full behavior of the model has been explored and checked against known and expected values. This completes the development and verification portion for Sustain-ME. Chapter 5 will delve into the full picture of what occurs in the sustainment process under these three maintenance paradigms.

CHAPTER V

EXPERIMENTS AND MAINTENANCE PARADIGM STUDY

Having created and verified Sustain-ME in Chapter 4, Chapter 5 will now demonstrate how such an environment may be used for performing sustainment decision trade studies. The methodology by which such a study is prepared is first defined. Next the steps of the methodology are performed, or if they have already been completed in other portions of this thesis those sections are referenced and the results summarized. Finally the information that can be gained from such a study is presented and explained in the context of a decision maker using quantifiable information to reach conclusions about a specific version or versions of sustainment.

5.1 Use Case and Use Methodology

The use case and example study for demonstrating Sustain-ME in this thesis were described in Chapter 1. Recall that the general use case referred to a decision maker attempting to justify or explore a specific decision. Sustain-ME enables decision making by translating information about the specific sustainment process being modeled into data on how the high level sustainment metrics evolve over time. Additionally, any information about how specific parameters within Sustain-ME change over time or about the frequency of different events within the simulation can be output to augment the high level metric information. In the case of this thesis, the fictional decision maker is attempting to determine which of three maintenance policies is the best, where "best" is defined by high R_O, a target value of 70% A_O, and ideally low

variability for both metrics. Aside from these high level metrics, the cost of the maintenance paradigm as captured by the level of initial inventory investment required to achieve a 70% level of A_O is an important parameter.

The fictional decision maker in this scenario is planning for future operations, meaning that she has some information about the expected reliability of the aircraft as well as the expected operational and supply chain structures. The order of maintenance activites is also well-understood, but there is uncertainty surrounding the capabilities of a new technology, the PHM, which will aid maintenance staff in predicting and diagnosing component failures. As a result the decision maker is aware that the maintenance strategies may perform differently as the assumptions about the PHM are varied. Thus one of her goals in using Sustain-ME is to quantify the regions where each maintenance paradigms is the best option, if there is such a region. Another goal is to determine the initial inventory investment level required to meet the A_O target for each of the maintenance paradigms under the most likely level of PHM effectiveness.

Before the decision maker is able to perform those studies, she must complete the methodology shown in Figure 87. As part of this methodology, specific questions about the modeling capabilities introduced to perform the trades must be answered. These include the questions posed in Chapter 1 from which Hypotheses 1 and 2 were developed; first whether the metrics of interest are both necessary to capture sustainment behavior, and second whether the prediction about stochasticity that motivated the study can be confirmed. The decision maker must also answer the question raised in Chapter 3 about what weighting parameters should be used for the CBM-MiMOSA optimizer. Finally, the decision maker must study CBM-MiMOSA in some depth before comparing it to other maintenance paradigms to ensure that the results gained are truly comparable.

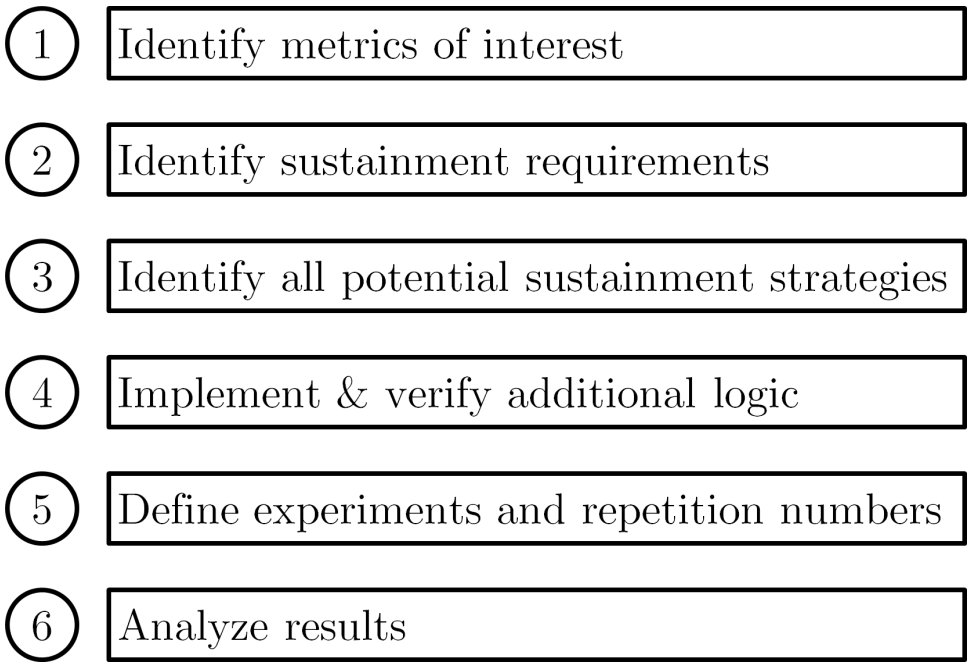


Figure 87: Use case methodology

Having fully captured the notional scenario for decision making that will demonstrate the utility of Sustain-ME, Section 5.2 will now describe how each of the steps in Figure f:UCM have been or are implemented.

5.2 Step 1: Metrics of Interest

The use case methodology's first step is to identify the metrics of interest. This was begun in Chapter 1 and continued in Chapter 2. The metrics chosen were operational availability and operational reliability, as well as the initial inventory investment as a surrogate for cost. However, Chapter 1 also recognized that A_O and R_O are closely related metrics and asked whether both must be measured by Sustain-ME to fully capture the sustainment behavior. Hypothesis 1 states that both are necessary.

Hypothesis 1: The relationship between A_O and R_O is complex and cannot be represented by a simple correlation.

This is the point at which the fictional decision maker should test such a hypothesis,

so that the correct metrics are identified before collecting any data. Section 3.2.1 described how this hypothesis would be tested: by varying the inventory with resonable resources and the resources with reasonable inventory. Hypothesis 1 will be supported if these two studies identify different relationships between A_O and R_O, particularly if a reversal in trend between the two occurs from one study to another. These studies were performed and the data was used to plot the correlation between A_O and R_O as inventory and other resources were varied. Figure 88 shows this relationship as the inventory was varied, and Figure 89 shows the relationship as the number of ground crews was varied.

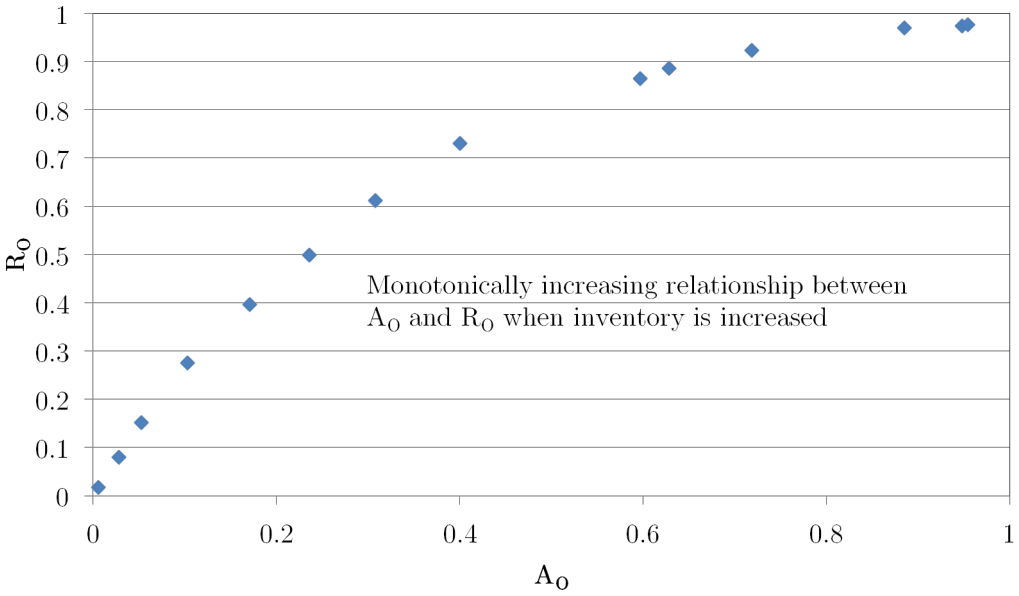


Figure 88: A_O vs. R_O for varying inventory

Figure 88 shows that the traditional and expected relationship between operational availability and reliability holds when the inventory is varied. Adding more spare parts to the system allows more aircraft to be available, and this in turn allows more of the required missions to be flown. The relationship is nonlinear, as the operational reliability nears its maximum faster than the operational availability, and therefore shows diminishing returns for higher A_O levels. This in part helps to explain why the Air Force might want to contract for lower A_O levels, because almost perfect

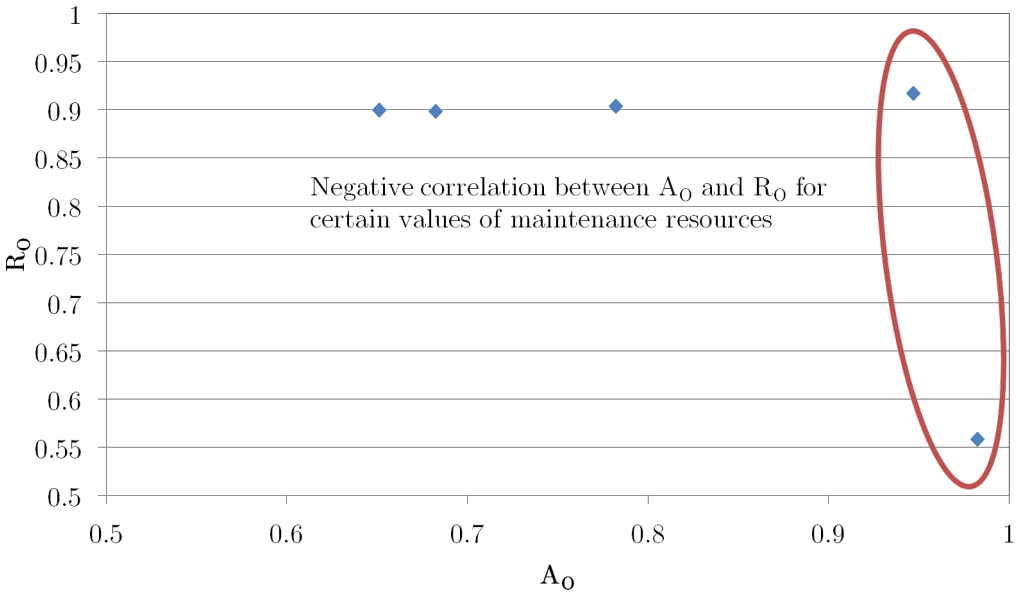


Figure 89: A_O vs. R_O for varying ground crew

ability to fly all required missions occurs at lower A_O values than 90%. However, it is important to remember that the points shown here represent the mean over time as well as over several repetitions, and therefore this neat relationship is subject to a lot of uncertainty.

Figure 89 does not quite show a reversal in trend. However, it does show that, for some conditions, high A_O does not necessarily translate to high R_O. In this case, having a single ground crew led to high operational availability because, as missions are generated, the aircraft must queue for ground crew in order to fly those missions. Because the prep time for missions without any wait for a ground crew takes a few hours, and the ground crew is required to be present for those few hours, aircraft that arrive later to the queue wind up waiting an unreasonable amount of time – in some cases as much as 50 hours. This in turn leads to low values for R_O, because required missions are not flown. However, A_O remains high because the aircraft are categorized as available during that time. Furthermore, because fewer missions are flown there are fewer opportunities to break and require repair, which reinforces the high A_O result. For this reason alone, it is clear that both A_O and R_O are necessary metrics

to capture and evaluate decisions against. This observation leads to the conclusion that **Hypothesis 1 is supported by Sustain-ME**.

Having determined that both metrics are necessary to any discussion of sustainment behavior, the decision maker may now proceed to identifying sustainment requirements.

5.3 Step 2: Sustainment Requirements

The use case methodology's second step is to identify the requirements of the sustainment process. This was performed throughout Chapters 1 through 3, as the sustainment requirements had to be defined to develop Sustain-ME[1]. Particularly, the expected activities associated with operations, maintenance and the supply chain were derived from literature, as well as the required metric values or directions of improvement. In this case, the desire to have 70% A_O and the maximum possible R_O and lowest possible initial inventory investment was determined from several government contracting sources.

In addition to these basic requirement definitions, Chapter 1 spent a great deal of time discussing the potential impact of stochasticicy on sustainment under the particular operational conditions associated with the Air Force sustainment paradigm shift. In particular, Hypothesis 2 predicted that nonstationary behavior will be observed for sustainment under CBM with minimal inventory investment.

Hypothesis 2: At the conditions cited in the Air Force sustainment paradigm shift, where minimal inventory is selected to meet a target value of 70% A_O, and where maintenance is performed based on a condition based maintenance policy, stochasticity will dominate the performance of sustainment.

[1]Now that Sustain-ME is developed, however, the particular sustainment requirements associated with whatever study is being performed with Sustain-ME must be defined independently.

As stated in Chapter 3, the test for this hypothesis will run Sustain-ME with a traditional CBM paradigm implemented over a wide range of inventory values. This was done and, from the data collected, a plot similar to Figure 59 was created to show the effect of inventory change on A_O. Each point on the plot represents the average of A_O over the course of a year of simulation time. Since the inventory investment took time to be used up, leading to an initial period of uncharacteristically high performance, the average was taken over the last 265 days of the year. Figure 90 shows the average operational availability as the total initial inventory investment across all the part categories is varied under traditional condition based maintenance.

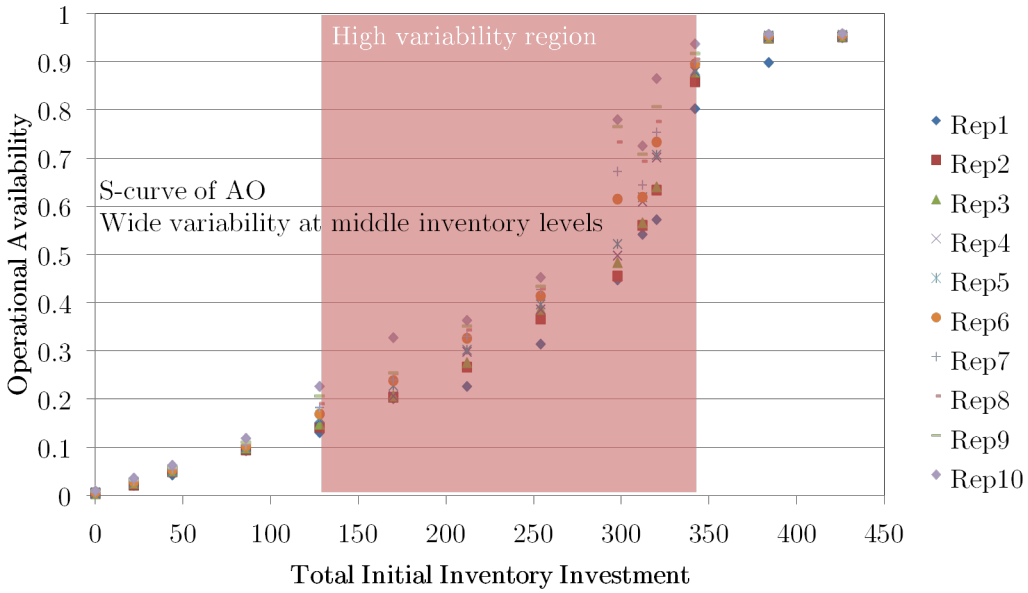


Figure 90: Inventory impact on A_O over full inventory range

As predicted, Figure 90 looks similar to Figure 59 from Chapter 4 in that the transition period between low and high A_O as inventory increases is characterized by a wide (vertical) range of time averaged A_O values. Figure 91 shows the same data, but focused on this medium range of inventory where the time average of A_O is close to 70%, as specified in Hypothesis 2. In this limited range, which represents the desired operating conditions for the Air Force sustainment paradigm shift, the range of A_O is significant, around 30%. This is problematic because it makes it difficult

to be certain that a targeted and contracted level of A_O can be achieved, since the circumstances have such a large effect on A_O.

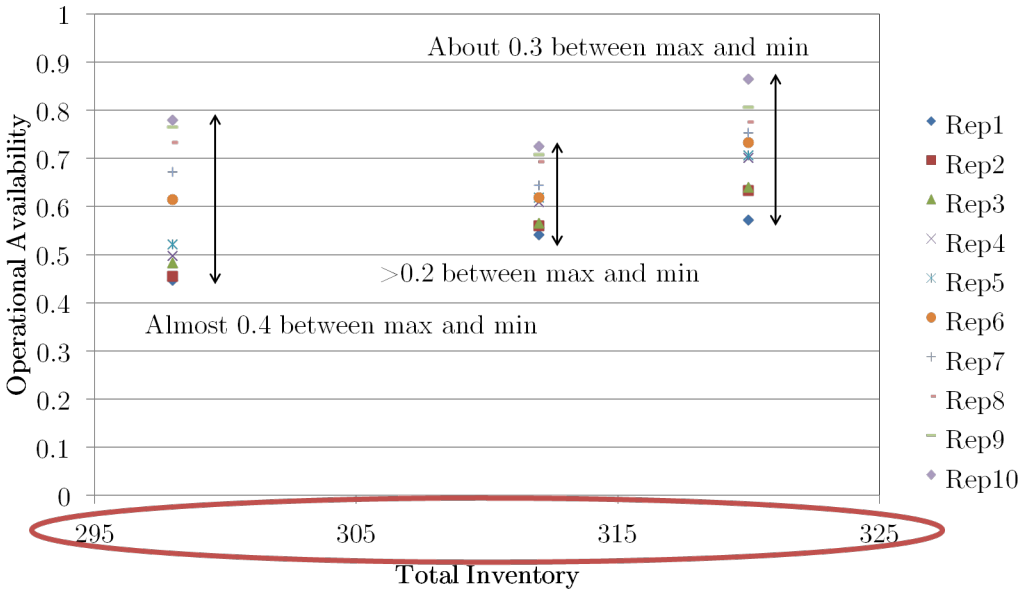


Figure 91: Inventory impact on A_O over select inventory range

Another test for sustainment nonstationarity under a CBM paradigm and minimal inventory is to look at the operational reliability (percent of missions flown) as inventory is varied. Figure 92 shows how the R_O varies with inventory. The overall behavior is the same, with the time averaged R_O increasing as inventory is increased, but the shape of the curve is slightly different. Where the A_O plot forms an S curve, the R_O plot is generally linear up to the point that it reaches a maximum and levels out. This suggests that, though the R_O also experiences increased variability for transitional inventory levels, it has a slightly more predictable behavior. Nonetheless, for the inventory level that achieved an average value of 70% A_O, 312, the spread in R_O between different repetitions is still a fairly large 10%. Though this is better, it is still not desirable. Moreover, the variability *within* a single repetition, as can be seen in Figures 93 (one model run) and 94 (all ten model runs), and Figures 95 (one model run) and 96 (all ten model runs), is much larger even than Figures 90 and 92 indicate. The effect of collapsing an entire time series of data into a single mean value

is to hide much of the variability that exists.

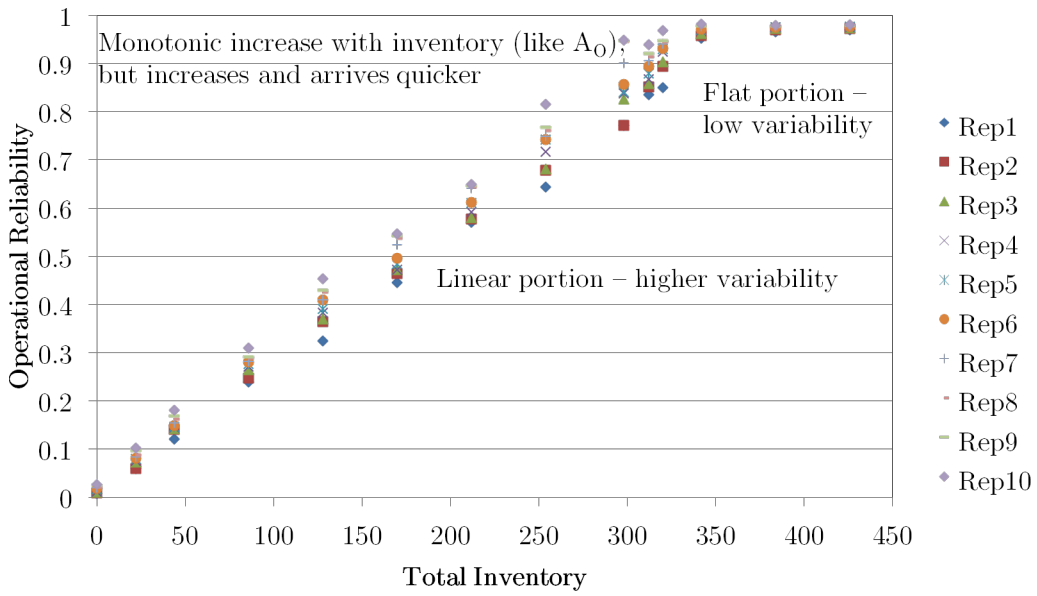


Figure 92: Inventory impact on R_O

Figures 90 through 96 clearly indicate that sustainment under condition based maintenance displays nonstationary behavior for inventory levels at which the A_O is 70%. This is concluded due to the large scale oscillations in the responses of A_O and R_O over time, the lack of any clear trend that emerges from these responses (i.e. a regular period of oscillation), and the inconsistency of the average of the responses over several repeated trials with the same inventory level. This is, of course, based on the specific assumptions made in creating Sustain-ME for this thesis. However, given the indicators in literature and the conceptual reasons for thinking this would be the case, it seems to indicate that this is a concerning region for anyone to be operating. This leads to the conclusion that **Hypothesis 2 is supported for the conditions modeled in this thesis**.

Having determined that nonstationary behavior does seem like a reasonable expectation for sustainment under the paradigm shift conditions, the decision maker may now select sustainment strategies with this fact in mind.

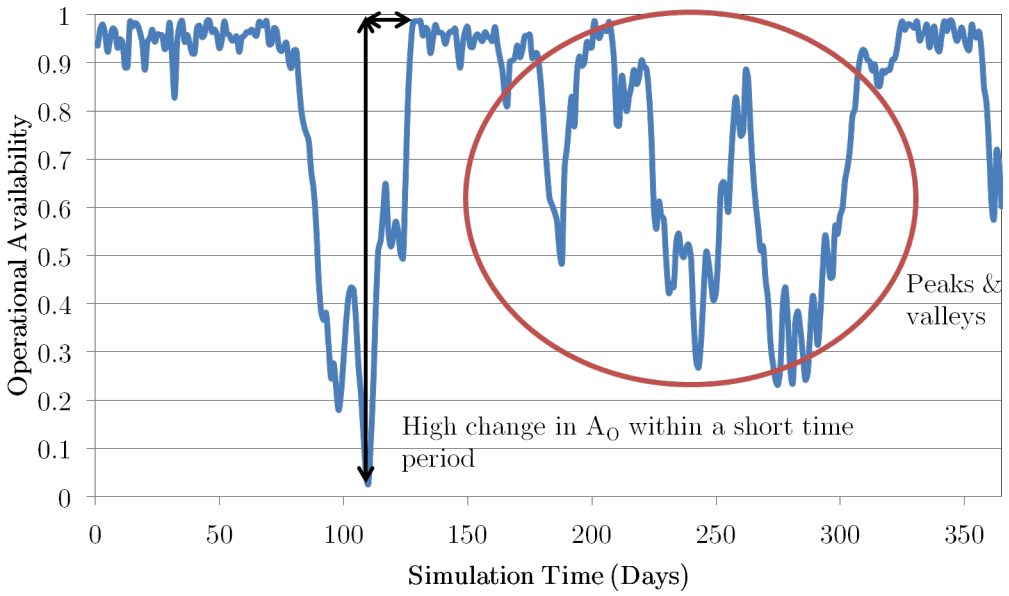


Figure 93: Single model run: A_O vs. time with 312 total spare parts

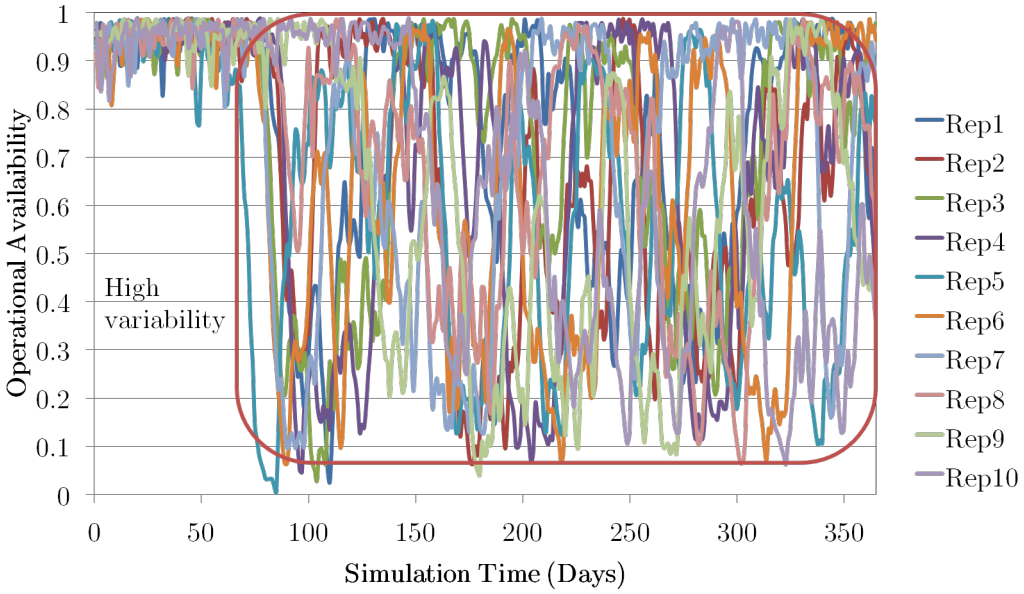


Figure 94: Ten model runs: operational availability (A_O) vs. time with 312 total spare parts

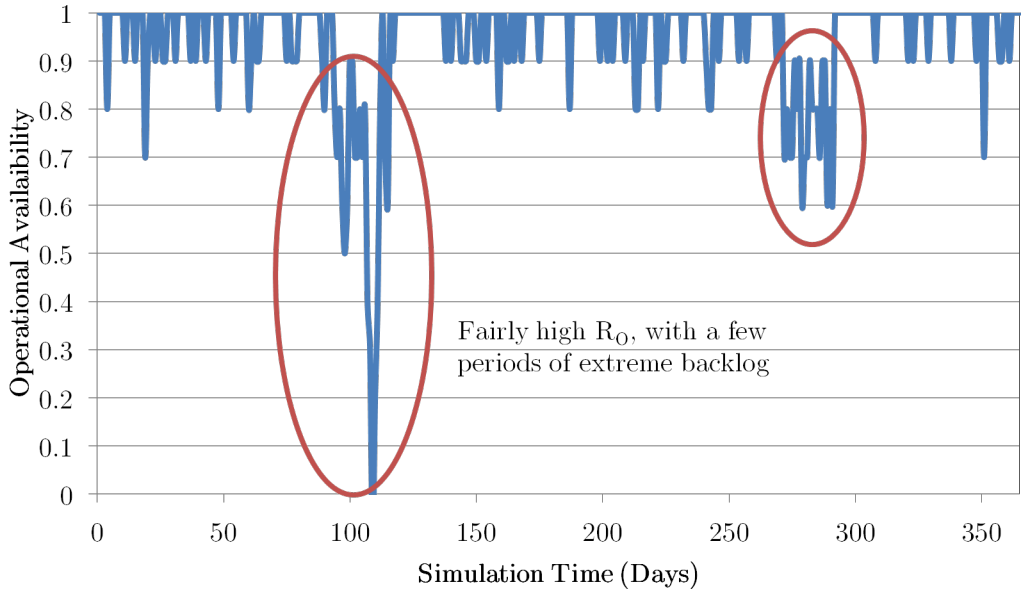


Figure 95: Single model run: operational reliability (R_O) vs. time with 312 total spare parts

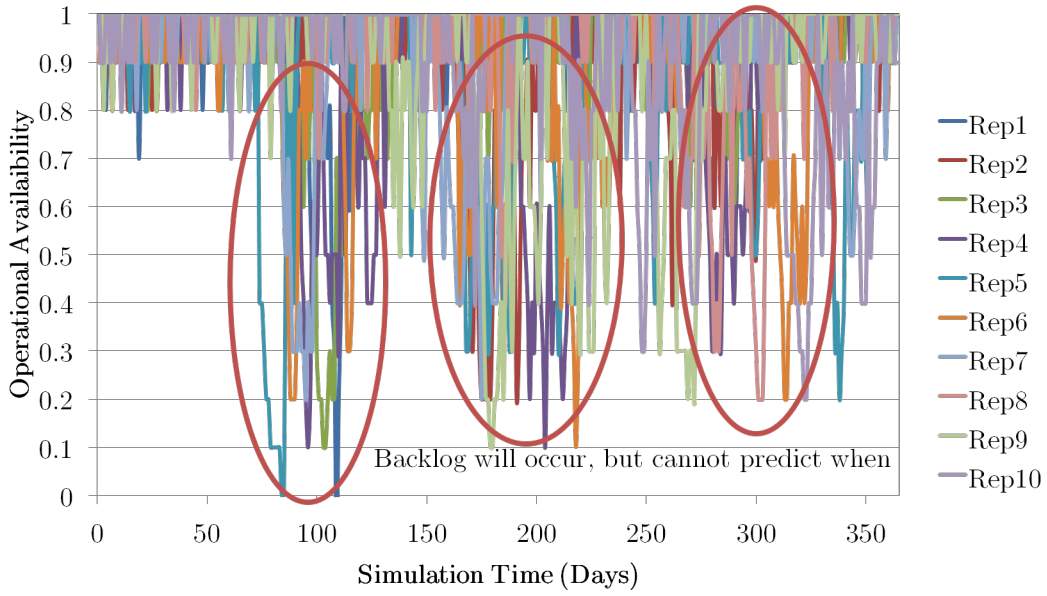


Figure 96: Ten model runs: R_O vs. time with 312 total spare parts

5.4 Step 3: Potential Sustainment Strategies

The use case methodology's third step is to identify potential sustainment strategies. This was done in Chapter 1 to some extent, but moreso in Chapters 2 and 3. In this case, due to the knowledge from Step 2 that traditional CBM is likely to create non-stationary behavior under conditions of minimal inventory, the decision maker would like to test the assumptions underlying traditional CBM regarding the decisions for when to schedule maintenance. She recognizes that the focus of CBM on maintaining as late as possible may be introducing new problems by allowing stochasticity to dominate maintenance scheduling. To determine whether a better option exists for using the PHM information that enables CBM, she conceptualizes an optimization problem that will relax the assumption that repair occurs at the last minute. Instead, the optimization will trade off the part life wasted when parts are repaired early against the benefit that may be gained from shifting maintenance visits to more even times. The decision maker also knows that CBM has not been modeled before, so its benefit as compared to a traditional maintenance paradigm like reactive maintenance is unknown. The decision maker therefore decides to study and compare three maintenance paradigms: reactive maintenance, traditional CBM and the novel CBM paradigm that she calls CBM-MiMOSA.

5.5 Step 4: Additional Logic

The use case methodology's fourth step is to implement and verify additional logic. Since this thesis built a full modeling environment to capture this logic, all logic needed to be verified; however, future efforts using Sustain-ME will only need to verify changes that are made to the model. This means that much of the implementation and verification for this use case example has been done in Chapter 4. However, two additional efforts must now be discussed. The decision maker is aware that the objective function that is used for the optimization portion of CBM-MiMOSA is based on

the values used to define the OEC. Particularly, the weights placed on the individual components of the overall objective function may influence the solution chosen. The decision maker must therefore perform an initial study to determine the appropriate weighting values. Additionally, the decision maker expects that the increase in part life waste associated with CBM-MiMOSA as compared to CBM will lead to a need for more maintenance resources; more initial inventory investment may or may not be required, but additional maintenance staff or facilities are expected to be required based on the fact that the total number of maintenance events will increase. Thus the decision maker will also perform a preliminary study of the performance of CBM-MiMOSA and determine what additional resources are required, if any.

5.5.1 OEC Weighting Study

The decision maker would like to determine the ideal values for the weightings on the individual objectives of the optimization problem. Since the objective functions were normalized, the weightings can be varied from 0 to 1 and should have equal impacts at those levels on the individual objectives. This is helpful because it means no objective can dominate the others. A full factorial design of experiments (DoE) was run on the weightings, where each weighting was varied in increments of 0.2. This test was performed first on the results from a single solution of the optimization problem. What this means is a representative set of optimization inputs was taken from Sustain-ME. These inputs are the decision variables of the objective function, which means the state of the fleet at a single moment in time from the simulation was selected as the test case for the optimization problem. For this test case, the weightings were varied and the optimization problem solutions were collected. The fleet conditions used for testing weight sensitivities vary from 10% to 100% of aircraft available. Since results from running the PHM version of Sustain-ME suggest that approximately 2/3 of aircraft detect failure of those which are available, this result was kept constant

as the number available was varied. To ensure these numbers, an initial fleet was generated from the simulation which had 20 aircraft available predicting failure, and 10 aircraft avilable predicting no failures. For the 90% available fleet, three aircraft were chosen at random from the initial fleet, two from the predicting failure portion and one from the not predicting failure portion. One of the parts for each of these aircraft was selected at random to be failed, although which part was failed would have no impact on the results of the optimization. This process was repeated for each additional percentage available fleet down to three aircraft available with two predicting failure and one not. Each of these fleet makeups was tested over a wide range of weightings on each of the three objective functions, with one weighting being adjusted at a time to determine the independent contribution of each sub function's weighting value.

Figure 97 shows the results of this DoE as a scatterplot matrix, which allows several individual scatterplots to be compiled together in a single graphic. This graphic can be read by looking at the input variable, along the x axis of the matrix, and the output variable along the y axis. In this case the input variables are the weightings from 0 to 1 in increments of 0.2, and the output variables are the actual values of the individual components of the objective function. Recall that the overall goal was to minimize the maintenance interval and part life wasted objectives and to maximize the missions flown. This translates to low values in the first two rows of the scatterplot matrix being preferable, and high values being preferable in the third row.

Figure 97 shows a few clear trends. If the top left to lower right diagonal of the matrix is viewed, these plots represent the impact of the weighting of an objective on the best value of that objective found by the optimizer. The trends for the two objectives (that are to be minimized) is negative as the weighting value increases, that is, the optimized objective function valule improves. The trend for the third objective

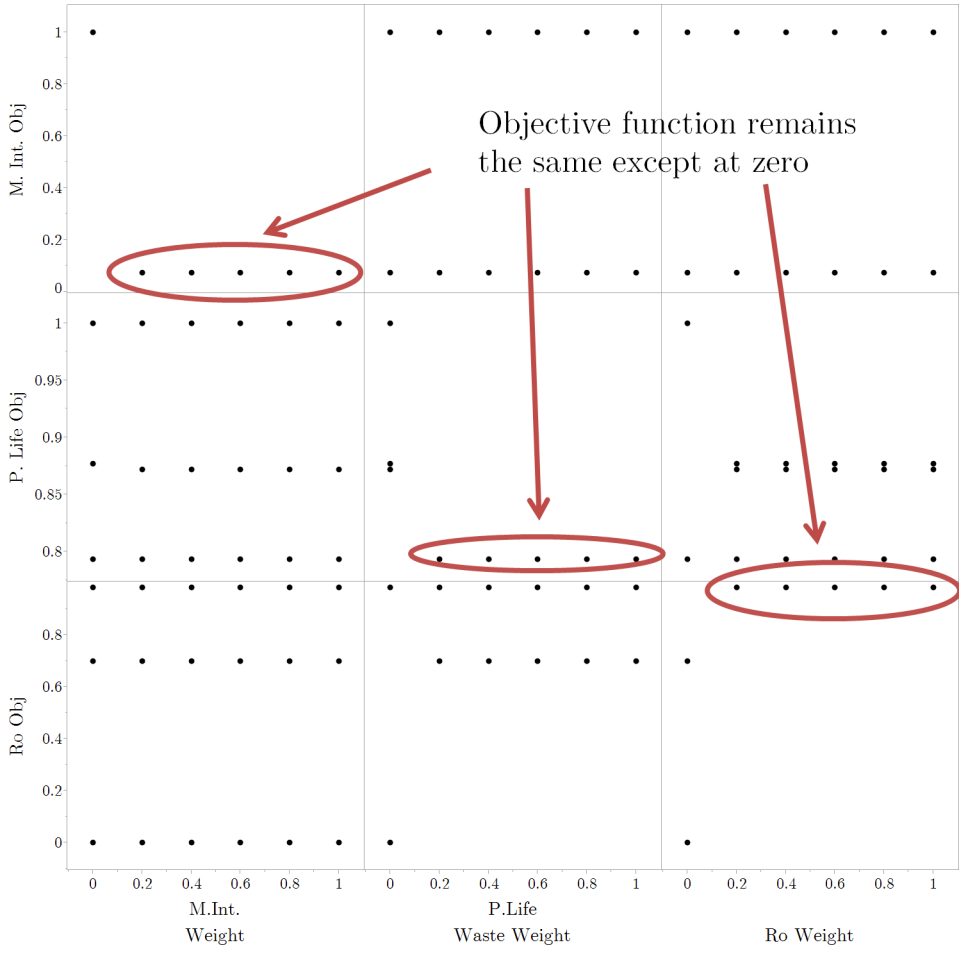


Figure 97: OEC weight study results – all weight settings

(that is to be maximized) is positive as the weighting value increases, indicating that this objective function also improves with weighting value. Given this fact, an interesting result emerges when the zero values are removed. Keep in mind that weighting a component of the objective function to zero implies that that objective is no longer being used to find a solution; doing so is like deciding not to optimize that value after all. Figure 98 shows that for all nonzero values of weightings, i.e. all reasonable weighting values, there is no impact of weightings on the optimization's solution. As a secondary observation, this means that all objectives are necessary to find the best overall solution, as a worse solution is found if any of them are zeroed out. A tertiary observation is that none of the objectives are correlated with one another for this particular optimization problem, indicating that none should be dominating the behavior of the response.

This further suggests that the optimization problem as constructed has a single optimum solution, with no ability to adjust the solution found by adjusting the weightings of the different portions of the objective function. This result naturally calls into question whether the solution has been improved at all, since it is possible for an optimizer to return the same solution that was obtained before optimizing. This can be formalized as a research question, since it is central to determining the usefulness of CBM-MiMOSA.

Research Question 3 Does CBM-MiMOSA improve the maintenance visit distribution over time?

To test this research question, a baseline solution to a momentary scheduling situation with the same fleet composition must be extracted from the simulation and compared to determine if the maintenance visit spacing has improved. This can be

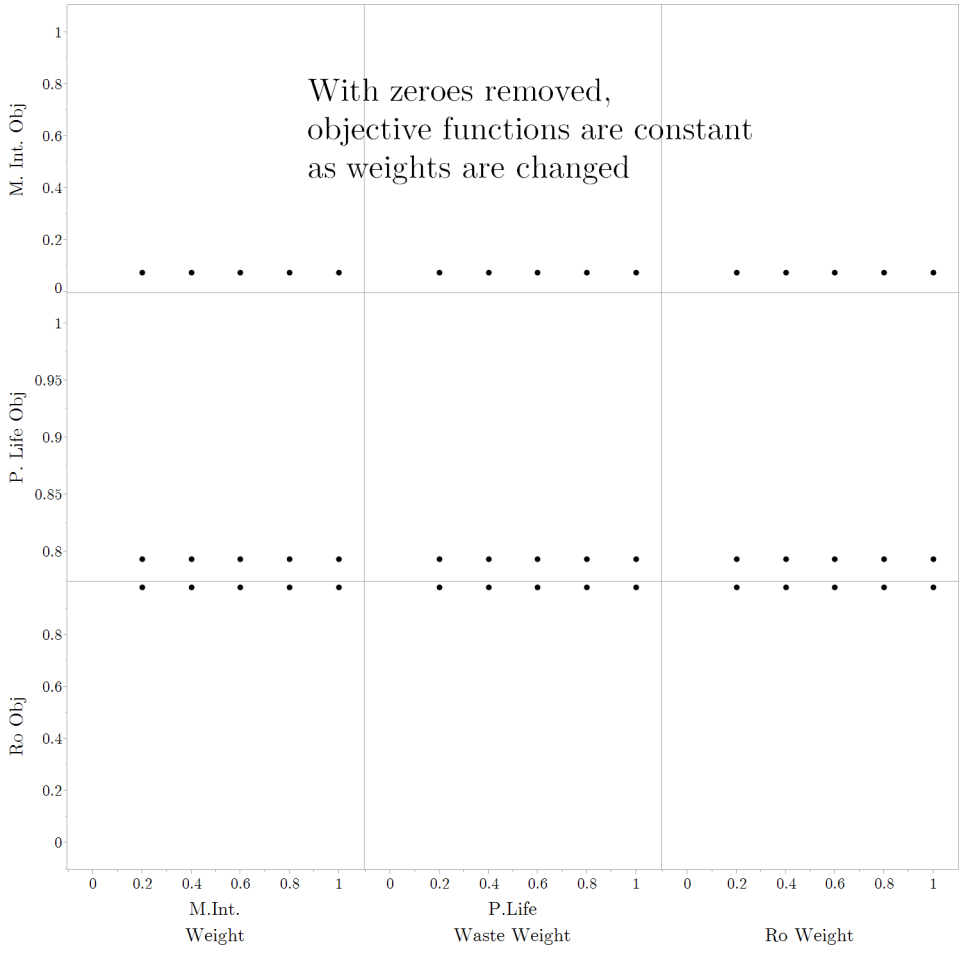


Figure 98: OEC weight study results – zero weightings excluded

done by looking at a number of parameters. First, the maximum of all the maintenance intervals over this period should be smaller under optimization, indicating that fewer periods of underutilized maintenance occur. Second, the minimum of all the maintenance intervals should be larger, indicating fewer times when multiple aircraft fail and visit maintenance at once. Finally, the standard deviation of the maintenance intervals should be smaller indicating that each one more closely matches the mean value.

To test this, the fleet with 30 aircraft available was used to allow as large a sample size of maintenance intervals as possible. The fleet was initialized with the same composition as was used in the optimization only test, and the maintenance visits over the same few day period were measured using the PHM model and the original baseline. The times at which aircraft enter maintenance for these models were collected and compared to the maintenance times determined by the optimizer.

Table 20: Minimum, maximum and standard deviation of maintenance intervals

	CBM-MiMOSA	CBM	Reactive
Min	7.20	0.23	0.39
Max	9.60	648.88	621.00
Standard Deviation	0.76	176.51	154.89

Table 20 shows that the spacing between maintenance visits of the optimized solution is *far* more even than for either of the other models, and this is reinforced in Figure 99 which plots the interval length between adjacent maintenance visits on a log scale on the y-axis versus the maintenance event order on the x-axis. This shows that the maintenance times vary by a much greater amount for the reactive and traditional CBM paradigms. However, the area under these curves indicate the time it takes to complete the full list of maintenance events, and this time is much shorter for CBM-MiMOSA. To see why this occurs, Figures 100 and 101 plot the times of each of the maintenance visits along the x-axis for the different operational paradigms (the y-axis values are only present to create spacing between series). However, two things

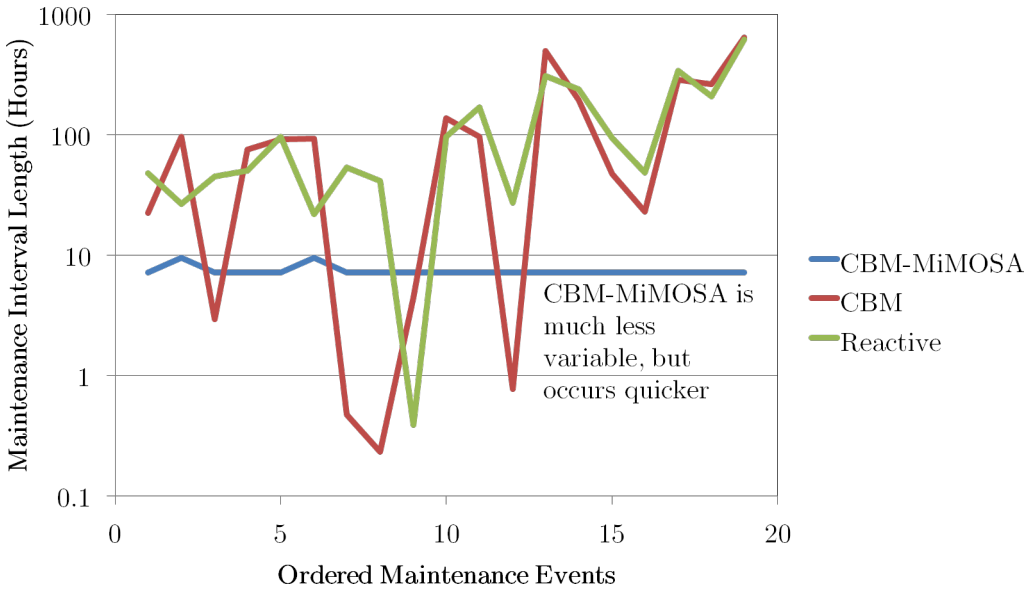


Figure 99: Interval length (log scale)

Figure 100: Maintenance visit spacing for optimized solution

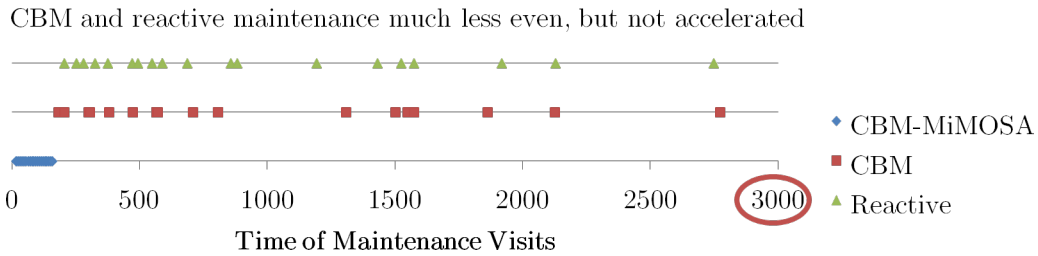


Figure 101: Maintenance visit spacing for CBM-MiMOSA, CBM alone, and reactive maintenance

must be pointed out. First, the period of time over which the optimizer performs the repairs is *much* shorter than the period over which the other two methods act, by a factor of 17. This indicates that, because of the formulation of the optimization, repairs are being made with a much different schedule than before. Though this was ostensibly the intention of CBM-MiMOSA, the degree to which part repair has been accelerated by the optimization suggests that part usage will dramatically increase, leading to a need for more parts at the very least. It was hoped that the optimization could instead shift maintenance visits around over the same or a similar period to what occurred before implementing optimization.

The second aspect of the problem to remember is that Figures 100 and 101 and Table 20 show only the repairs that directly correspond to those parts which had planned repairs within the one instance of the optimization problem. A more direct analogy might be to show *all* the maintenance visits that occurred over this period, shown in Table 21 and Figure 102. These confirm that, though the spacing is much more even when accounting for all repairs that occur over this period, it still falls far short of the maintenance spacing achieved by a single run of the optimization. Thus even though the optimization is insensitive to weightings, suggesting that the problem may have been overly constrained in its formulation, and even though it seemingly creates many more maintenance events than existed for the baseline models, it does still represent an improvement in the intended parameter over the baseline scheduling methods. Furthermore, though the optimization formulation is overly constrained, it does suggest that the assumptions made in creating that formulation help to achieve the primary intended goal for the method, to evenly distribute maintenance visits over time. However, the fact that the visits are compressed over a small time period may mean that the assumptions made reduced the freedom of the optimizer to reduce the penalty associated with this goal. This will be discussed further in Section 5.5.2.

To round out the analysis of the optimization problem in isolation, the data will

Table 21: Minimum, maximum and standard deviation of all intervals

	CBM-MiMOSA	CBM	Reactive
Min	7.20	0.23	0.24
Max	9.60	359.35	189.73
Standard Deviation	0.76	60.85	41.96

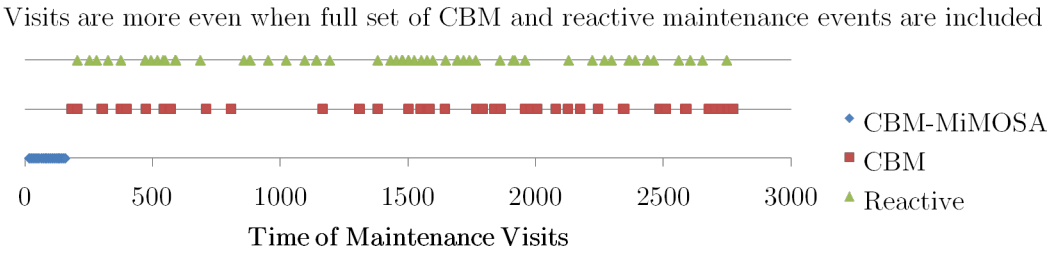


Figure 102: Maintenance visit spacing for CBM-MiMOSA, CBM alone, and reactive maintenance

also be examined to determine at what fleet availability percentage the fleet begins to have difficulty flying 100% of missions during the optimization period. This relationship is similar to the one that relates operational ability to operational reliability, although operational availability has a slightly different computation from the percent of aircraft available at any given time. In Figure 103, the missions flown only stay at 100% from 30 to 27 aircraft available; below this point the optimization cannot satisfy mission requirements over the entire operational period. This is due to the fact that 2/3 of the aircraft must be repaired and are therefore not available to fly after a certain day of the optimization period. In the greater simulation, the number of aircraft needed simultaneously to keep flying all missions may be lower than this number due to the fact that maintained aircraft will be repaired and re-enter service, which does not happen in the optimization problem. However, depending on the number of maintenance resources and parts available at any given moment, this factor could cease to be relevant. Combined, these facts suggest that there is a dangerous potential to drive the operations of the fleet to a far inferior region by creating more maintenance events, though with more even spacing, than the current resources of the simulation can handle. Should this happen, the optimization will begin to fly

fewer missions due to the smaller number of aircraft available to fly them, which will in turn impact the rate of utilization of aircraft and the evenness of maintenance visits. These possibilities will be fully explored within the simulation later.

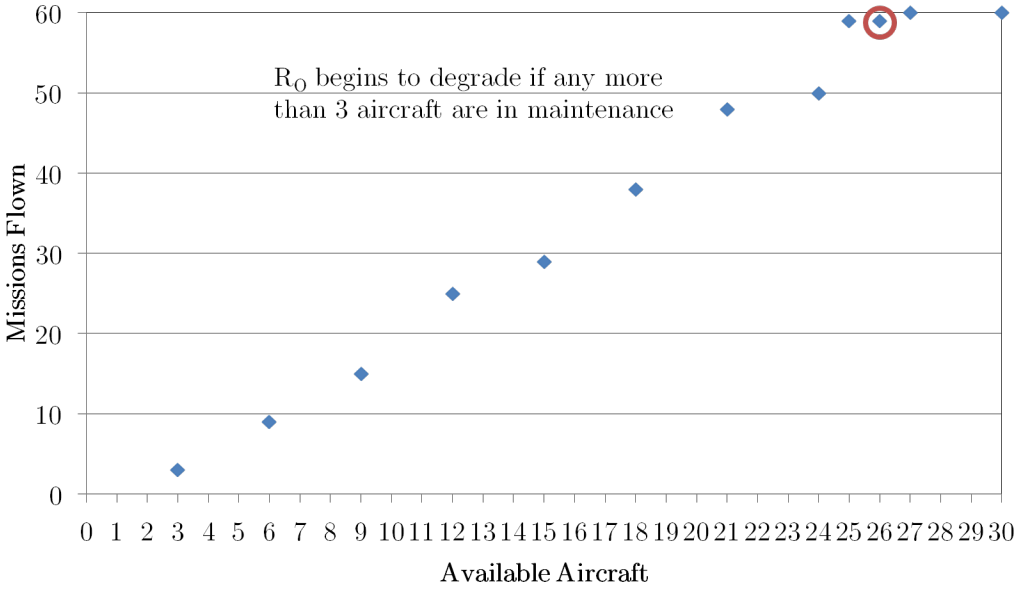


Figure 103: Effect on aircraft available on missions flown

Having identified that the OEC weightings do not influence the optimization solution found, the decision maker can move forward with the knowledge that any weighting values will suffice. However, the more important conclusion that may be reached from this study is that the optimization problem defined in Chapter 3 may not be the best option for achieving the decision maker's goal. Having examined the data further and found that the optimization problem does in fact yield a more even set of maintenance events, but that these maintenance events occur in a shorter amount of time[2], the decision maker decides to move forward with the comparison study to see what impact CBM-MiMOSA has on sustainment.

[2]Note that more even maintenance events over a shorter time period was the intended goal of CBM-MiMOSA, but that the degree by which the maintenance events are accelerated my overturn any benefit gained from reducing the stochastically occurring maintenance events.

5.5.2 Maintenance Resource Study

The decision maker would like to determine the impact of the maintenance scheduling rules of CBM-MiMOSA on the performance of sustainment to see if additional maintenance resources are required to support the new maintenance paradigm. Since CBM is the expected maintenance strategy under the Air Force's paradigm shift, and since CBM-MiMOSA is an attempt to modify the behavior of CBM, the decision maker will compare the two paradigms to see what changes in A_O and R_O occur under the altered CBM logic. To begin to tell the story of how these maintenance paradigms compare with each other as modeled, a single aggregated number was pulled from the simulation which collected data either every simulation day or every time a specific event occurred, with ten repetitions of Sustain-ME being run for each of the experimental settings. This single representative number first takes the average for a given repeated run of the metric over the simulation time. It next averages these averages to determine a broad approximation for the effect of these settings. The full simulation data contains much more information and will be presented later, but the aggregate allows a quick comparison between the different maintenance approaches. Figures 104 through 106 show specific aggregated parameters for each of the models, where the labels along the x-axis correspond to the different metrics. The y-axis gives the average of the average values for that metric, and where there are error bars these show the full range of the time-averages from the ten repetitions. Finally, the colors in the legend correspond to the two different models.

Figure 104 reveals that both operational availability and operational reliability are higher for CBM-MiMOSA in isolation. However, the range bars show that there is a great deal of overlap between the observed time averaged operational availabilities of the two, suggesting that on any two randomly selected datasets for A_O and R_O may show that either of the paradigms has the highest operational availability. However, the most significant goal for CBM-MiMOSA was to reduce the variability of the

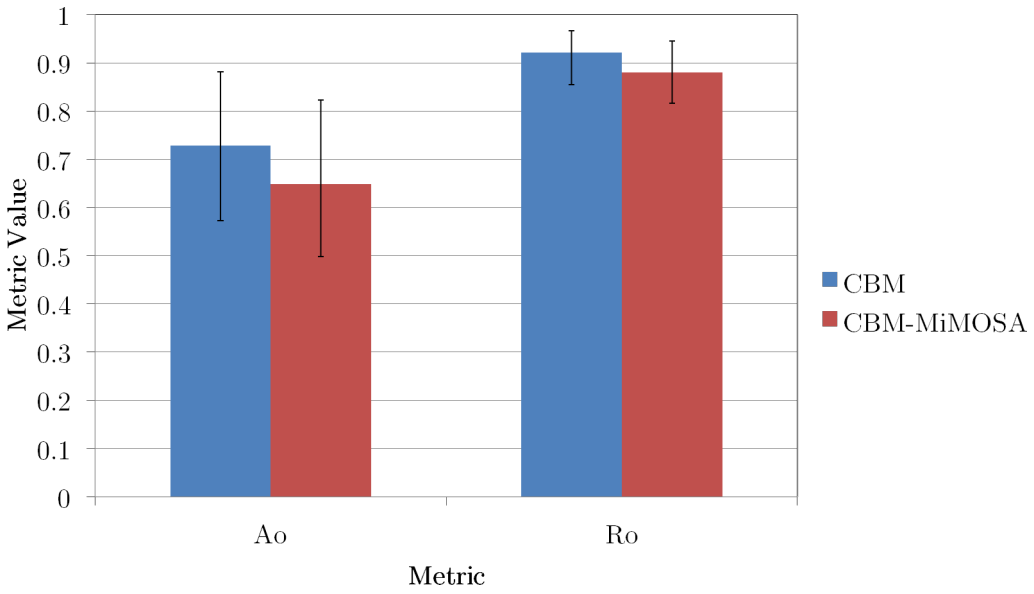


Figure 104: Average A_O and R_O for two maintenance paradigms

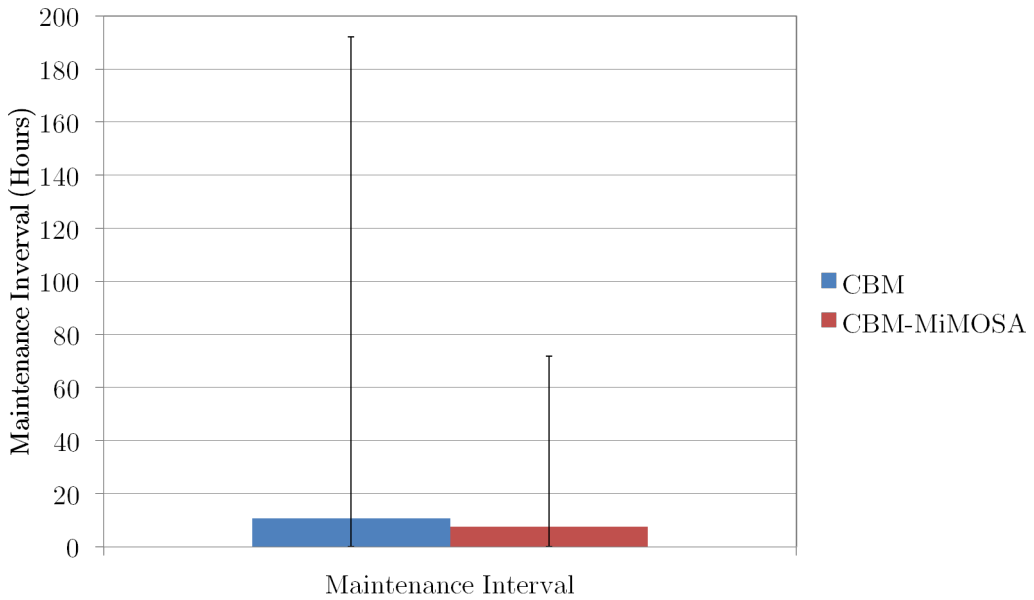


Figure 105: Average maintenance interval for two maintenance paradigms

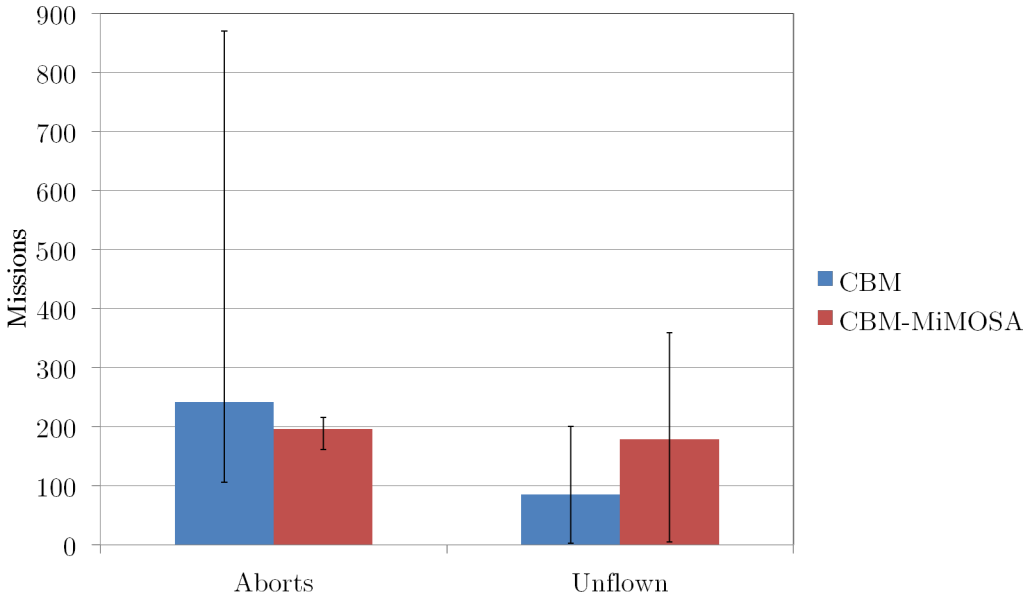


Figure 106: Average mission aborts and unflown missions for two maintenance paradigms

operational availability, not to necessarily increase the average. At the moment, Figure 104 suggests that this goal has not been achieved. Further tests will determine whether there is a region of operations or PHM detection lead times where this would begin to be satisfied. Figure 104 also shows that, despite occasionally having an Ao as low as 50%, the optimization maintenance method manages to keep an average operational reliability above 80%.

Figure 105 shows first of all that the distribution around the maintenance interval length is highly skewed for each of the maintenance paradigms due to fact that the maintenance interval is naturally bounded by zero. The high variability for both paradigms is due to the fact that, at 70% A_O, there are large periods of time with no maintenance events due to the wait for parts. However, both the mean and the total range are lower for CBM-MiMOSA, confirming that this method is at least successful in improving the evenness of maintenance visits. However, with a fairly large standard deviation even still, it would be desirable to improve this value even more, perhaps for different PHM detection lead times.

Figure 106 shows the relative frequency of missions which are unflown due to aborts (where parts fail during the mission) and missions which are unflown due to aircraft being unavailable. The number of aborted missions decreases from ALIS enabled maintenance to schedule optimized maintenance. It is not initially clear why there would be a difference in mission aborts between CBM and the optimized CBM approach. On the surface, these two methods have the same chance of seeing parts fail which are not detected in time to repair early. The true reason for this is more interesting, and will be explored in Section 5.7.1.

One further observation is relevant from Figure 106, and this is that the number of unflown missions increases between the CBM maintenance paradigm and CBM-MiMOSA. The more interesting fact is shown in Figure 107, that the total number of missions not flown during sustainment is greater for the optimized maintenance version. This shows that the optimization method actually reduces some of the benefit of the PHM for these particular settings. To explain why this is the case, we return again to the maintenance visits.

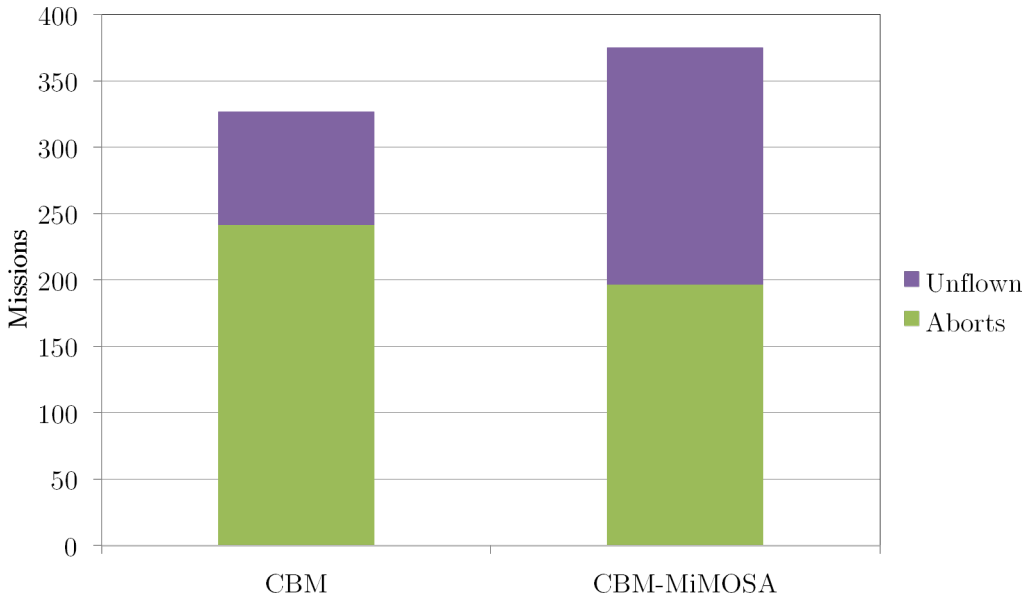


Figure 107: Average mission aborts and unflown missions for two maintenance paradigms

In Figure 105, only the interval between adjacent maintenance visits was of interest. However, the *number* of maintenance visits tells a bigger story: over the same simulation period and across ten repeated runs, Sustain-ME under optimized maintenance consistently had more maintenance visits, averaging 1164 visits compared to CBM-MiMOSA's 830, or 30% more. As mentioned previously, this occurs because the optimization method accelerates the schedule of maintenance events, replacing parts early. Over time, the missing part life builds into a greater number of parts replaced, which in turn requires more maintenance events. Thus the larger number of unflown missions due to unavailable aircraft for CBM-MiMOSA: aircraft were less frequently available because they were being maintained.

Another way to show this is to look at the percent of time spent in different states for each of the models. These categories are really just subcategories of the greater classifications "uptime" and "downtime" which are used to compute operational availability. The two "uptime" categories are available (meaning able to fly a mission and waiting for one) and flying. The three "downtime" categories are broken awaiting repair, broken awaiting parts, and broken and being repaired. Broken awaiting repair is also commonly referred to as non mission capable due to maintenance, and broken awaiting parts is commonly referred to as non mission capable due to supply[69]. The military distinguishes between these categories because they imply very different solutions when A_O is low, so it is helpful to track the time spent in each state separately.

Figure 108 shows the time spent in each category across ten repetitions for the PHM and optimized versions of Sustain-ME. The y-axis shows the percent of time spent in each of the five states on the x-axis. The error bars again show the range of values that the average took on over the ten repetitions. Figure 108 reiterates that the optimized version of Sustain-ME is available less frequently, but interestingly the percent of time spent *flying* over this period is barely lower. This does not mean

that the percentage is statistically insignificant, as the percentage itself is small and a small difference might still amount to a noticeably different amount of time spent flying between the two models. However, it does indicate that most of the operational availability reduction for CBM-MiMOSA can be attributed to lost time available, i.e. sitting around waiting for a mission, as opposed to actually flying missions.

Figure 108 also shows that this extra time spent as non mission capable can be attributed to the broken awaiting repair and broken being repaired categories; broken awaiting parts is unaffected. This indicates that aircraft under CBM-MiMOSA are more often present in maintenance, but that when they are there parts are on hand to fix them at the same rate as for the CBM version of Sustain-ME. This can be believed since it was one of the core ideas to support the idea of maintenance scheduling optimization: when parts are regularly utilized they are also regularly resupplied.

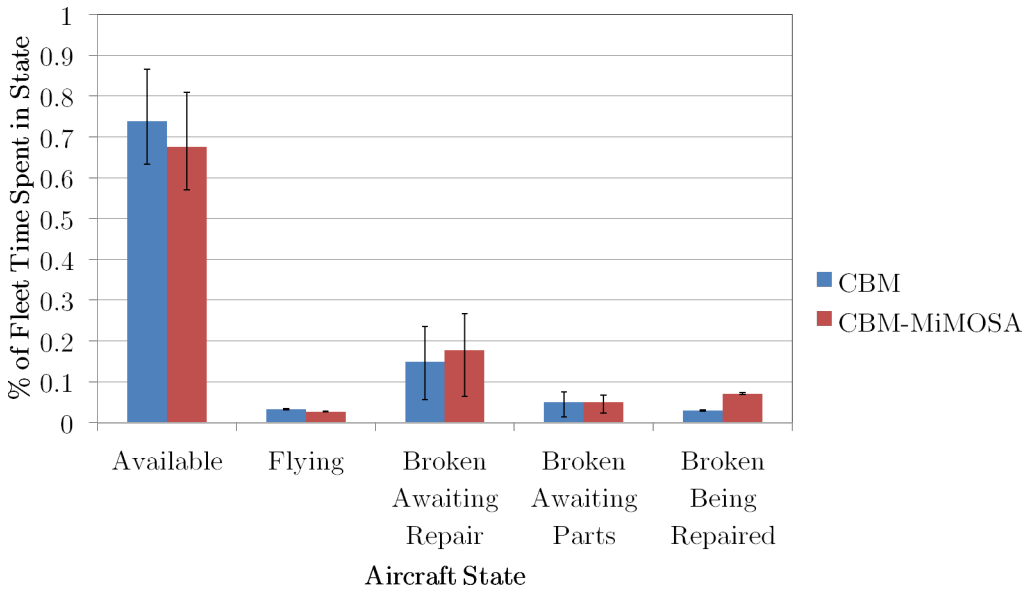


Figure 108: Percent of time spent in different states

To conclude this intial investigation into the relative behavior of the two models, the CBM paradigm alone outperforms CBM-MiMOSA. Due to the greater number of maintenance events required to even out the maintenance visits, CBM-MiMOSA was never able to overcome the disadvantages it introduced through further improvements,

except for the marginal case of a low PHM detection lead time. If the missions being flown by the two models were the same, the drop in A_O might be acceptable, but in this case both metrics suffer and this suggests that it might be worthwhile to infuse more maintenance resources into the optimized version of Sustain-ME to help reduce the time spent waiting for maintenance. However, if the additional resources have the same impact on the CBM paradigm as they do on the CBM-MiMOSA paradigm, this would suggest that one might as well invest the resources into CBM. Figures 109 through 111 explore this question.

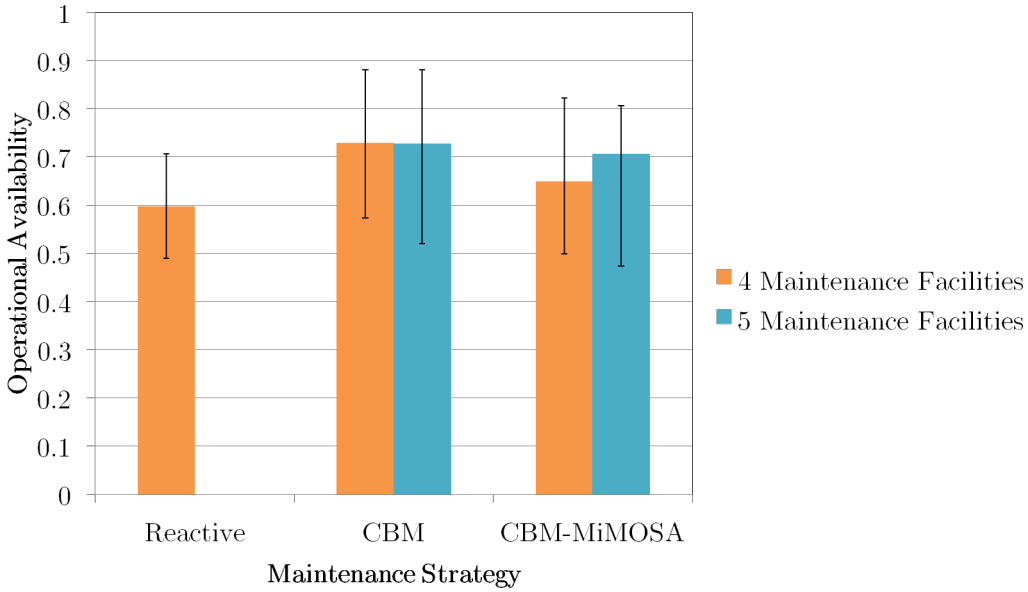


Figure 109: Average operational availability for three maintenance paradigms and with additional maintenance resources

From a macro level performance metric perspective, Figures 109 and 110 show that the additional maintenance facility does indeed improve the optimized maintenance model where the CBM version stays the same. Both A_O and R_O are almost idential for the CBM version after injecting more resources. This suggests that the additional maintenance visits cause a degradation in the optimization version which can be offset by investing in slightly more resources. The caveat of doing so is that the savings or benefit provided by the method must exceed the additional resources required to

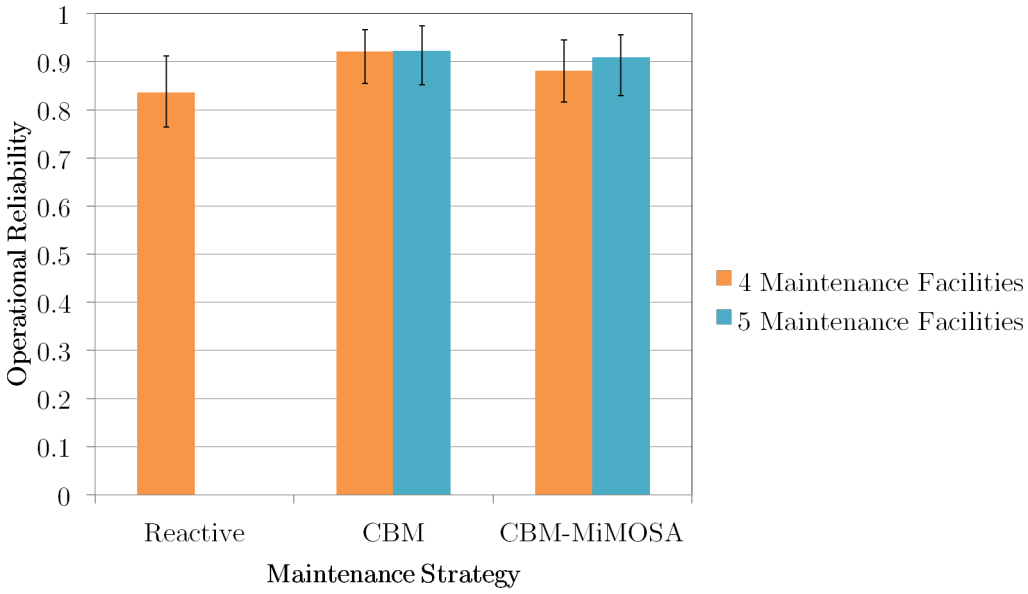


Figure 110: Average operational reliability for three maintenance paradigms and with additional maintenance resources

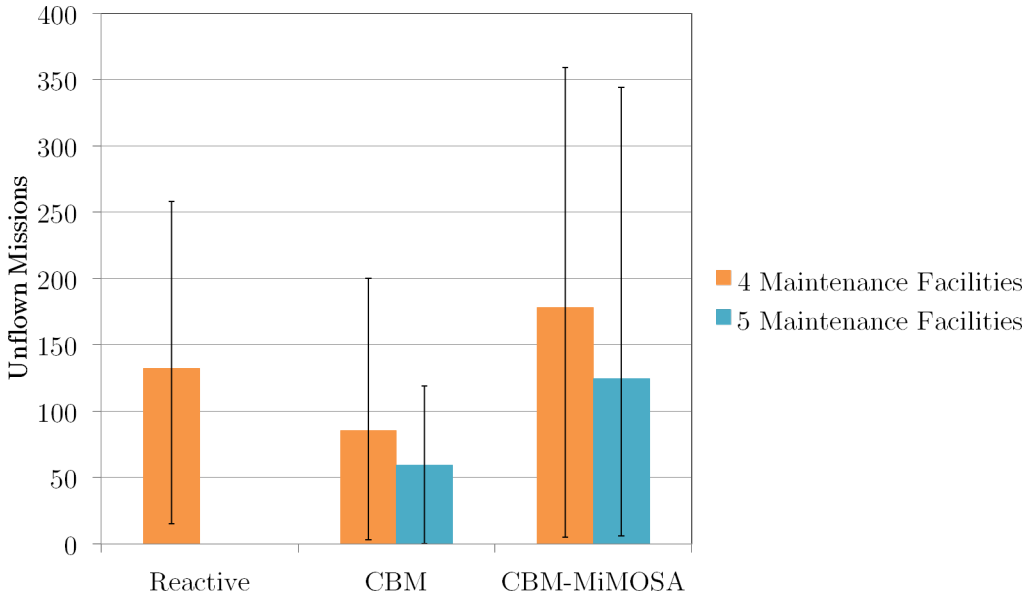


Figure 111: Average unflown missions for three maintenance paradigms and with additional maintenance resources

support it, but this should be assessed after the fact rather than prematurely imposing a penalty on the optimization method such that it never shows benefit.

Figure 111 shows the additional effect of prodivding more maintenance resources: both versions of maintenance have fewer missions which are unflown due to no aircraft being available, since the additional maintenance facility makes it less likely that aircraft spend downtime waiting for a facility to become available. The aircraft which are processed sooner can fly additional missions. However, it is important to note that the impact is once again greater for the optimized maintenance paradigm, indicating that the resource injection benefits one more than the other.

The decision maker can now justify why CBM-MiMOSA should be given slightly more maintenance resources than the baseline methods. At this point she is ready to define experiments to perform the main study about the three maintenance paradigms.

5.6 Step 5: Define Experiments

The use case methodology's fifth step is to define the experiments that will be used to perform the study, and to determine the number of repetitions needed to capture an accurate picture of sustainment. Due to the stochastic nature of sustainment, one data point from a combination of parameters will usually not be enough to determine the true behavior. In this case, the decision maker is interested in determining the PHM parameters for which different maintenance paradigms perform best, and in the initial inventory investment required to achieve 70% A_O for each paradigm. Therefore the first experiment will vary Sustain-ME's PHM parameter, detection lead time, over a reasonable range of values from 70% to 95% of part life to see how the behavior of the two CBM-based maintenance paradigms changes. Both CBM-based model paradigms will be given the same initial inventory investment so the two are comparable. For this experiment, the traditional reactive maintenance paradigm is not included because it does not use PHM information. Recall that the detection time defines the center

of the detection distribution, which translates into the amount of warning time with which the failure of a part can be detected before it needs to be replaced. Both CBM paradigms ought to become less effective as the detection time is decreased, because the amount of time available to react to upcoming maintenance events is reduced. However, quantifying this should provide useful information to real world engineers in creating real PHM systems.

The second experiment will vary Sustain-ME's initial inventory investment for all three maintenance paradigms to try to match a time averaged A_O value of 70% and compare the three inventory levels to see which maintenance paradigm is able to most efficiently achieve the requirement. This will be done for the 95% value of PHM detection lead time, which is expected to be a somewhat realistic value. In this case the experimental parameters are not known, since it is the goal (70% A_O) that is set. However, the initial inventory investment values are not expected to be extremely variable between the three paradigms.

For both experiments, the main metrics are A_O, R_O, and possibly inventory levels. However, other parameters from within the sustainment process may also need to be examined and compared to provide additional insight into the processes occurring. In this way a true picture of the effects of the three maintenance paradigms can be developed, rather than a superficial understanding based only on high level parameters.

The final element the decision maker must choose before performing the experiments is the number of repetitions required to capture the stochasticity within Sustain-ME. In this case, an experiment was done to determine the behavior of the mean over all repetitions as additional repetitions are added. Figure 112 shows how the mean has essentially stabilized after four repetitions, but that after nine a high degree of confidence in the mean can be held. As a result, ten was chosen as a round

number that should provide sufficient coverage of the stochasticity. This is additionally expected to be a sufficient number of repetitions because the test was performed at the most stochastic region, with inventory close to the value required to achieve 70% A_O.

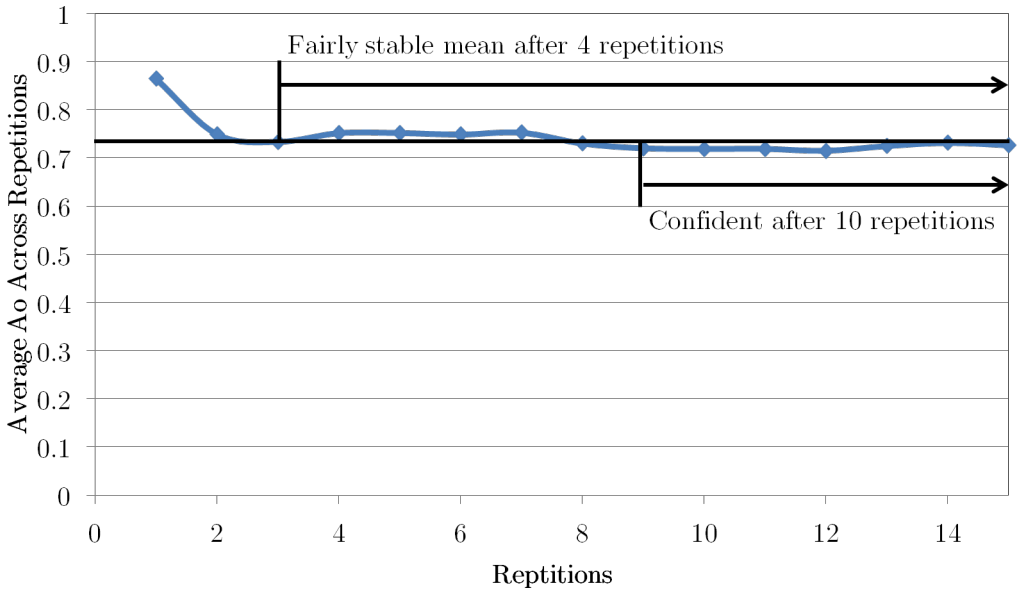


Figure 112: Convergence of A_O repetition average vs. number of repetitions

Having defined the experiments necessary for a comparison study of the three maintenance paradigms, the decision maker can now perform those experiments and analyze the results.

5.7 Step 6: Analyze Results

The use case methodology's sixth step is to perform the experiments defined in Step 5 and analyze the results. In this case, the two experiments will vary the PHM detection lead time for constant initial inventory investment (Section 5.7.1) and will vary the initial inventory investment for constant PHM detection lead time (Section 5.8). The designer will analyze the results of both experiments to determine if any maintenance paradigm clearly outperforms the others, and if so under what conditions.

5.7.1 PHM Detection Lead Time Study

As was stated in Chapter 3, the detection time parameter of the prognosic health management is expected to affect the behavior of the two CBM maintenance paradigms. For longer detection times, these paradigms should have more ability to react to upcoming maintenance events, as well as having less chance of detecting failure so late that a mission must be aborted. While the CBM maintenance paradigm has only one option for reacting to upcoming failures (it replaces the part on the last mission before failure will occur), CBM-MiMOSA can maintain the aircraft earlier if the optimizer suggests this would lead to a better solution to the objectives. This suggests that CBM-MiMOSA should see even greater improvement than the CBM paradigm for long PHM detection lead times. However, this was not found to be the case. Figure 113 shows that CBM-MiMOSA demonstrably *improved* as the detection time decreased (i.e. occurred at a greater percent of the part life), where the results for traditional CBM were less clear but if anything exhibited a downward trend in average and an increase in variability. Figure 114 shows that, though the A_O grows for CBM-MiMOSA as detection time decreases, meaning that the aircraft spends more time available, the R_O peaks at 80% and shows lower variability at 90%, suggesting that CBM-MiMOSA actually performs best at these detection time ranges.

It should be noted that CBM-MiMOSA has a higher A_O than the unscheduled maintenance paradigm at every detection lead time level except the earliest detection case. This suggests that it is a promising maintenance strategy in this metric, especially since it outperforms the CBM paradigm for PHM detection times of 90% of the part life or greater. Given that military aircraft are utilized in such a way as to make these detection times seem reasonable, CBM-MiMOSA shows merit. However, Figure 114 shows that, though the A_O of CBM-MiMOSA is better at these values, the R_O of CBM is still better. In this case the decision maker would choose the option that allows them to fly more missions, as opposed to the option that ensures aircraft

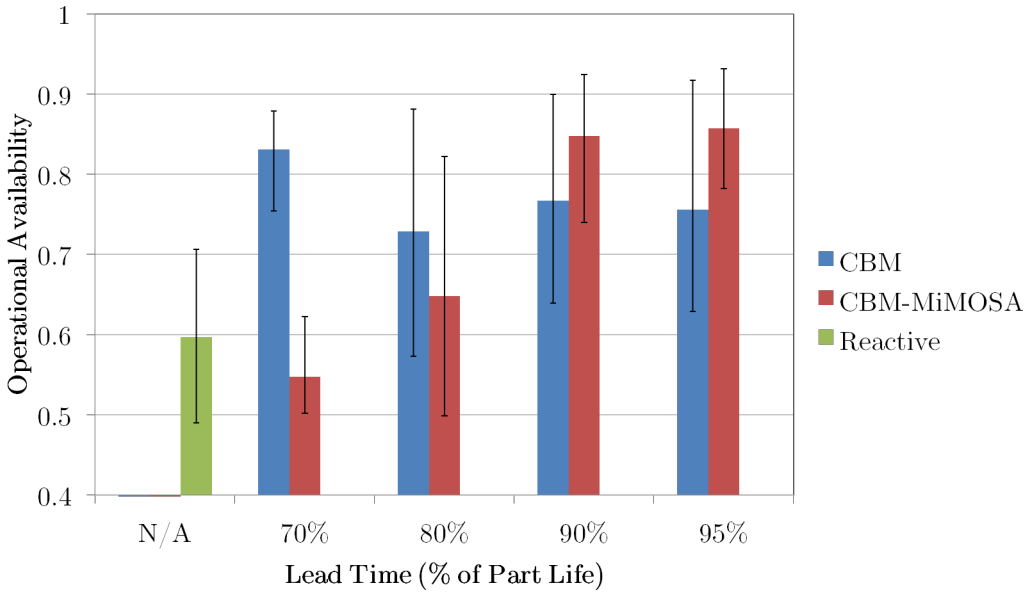


Figure 113: Effect of PHM detection lead time on operational availability

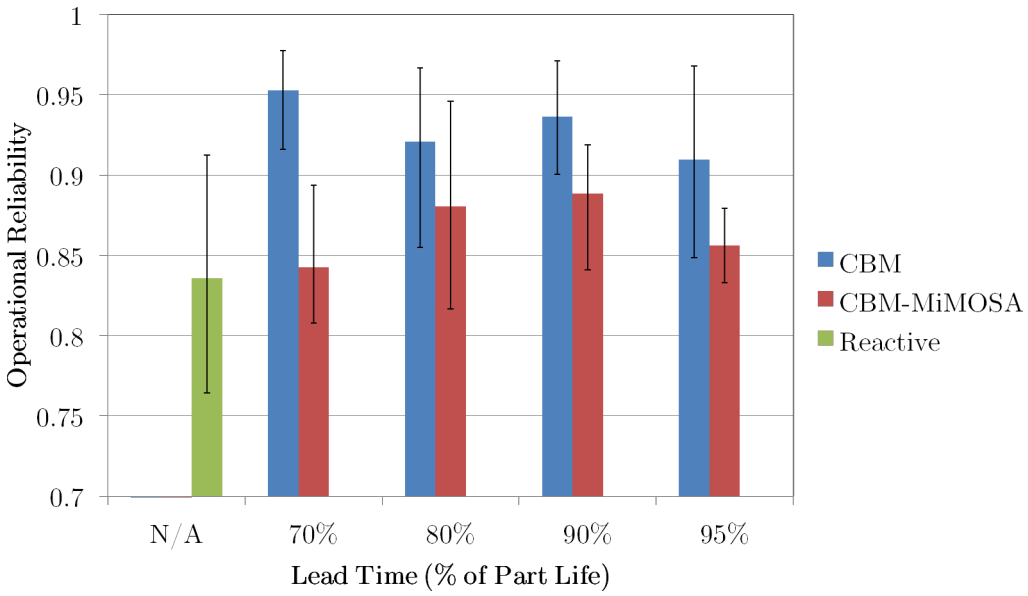


Figure 114: Effect of PHM detection lead time on operational reliability

are more often available to fly missions.

However, observing these results does not provide nearly as much information as delving into why they occur. As was mentioned in Section 5.5.2, the formulation of the optimization problem and the need to have a period over which to optimize led to the fleet's detectable maintenance actions being scheduled for maintenance within a few days of their detection, regardless of the amount of part life remaining. This decision was made because the mean time between failures for the fleet dictates that, on average, these maintenance events will occur within a few days. However, due to the degree of variation in when maintenance events occur, forcing the events to be maintained at the average interval leads to the acceleration of the pace of maintenance. The acceleration leads to a greater number of maintenance events occurring, which was expected, but at a much greater pace than was anticipated. On the whole, this means that when detection times are long, the optimizer's mandate to repair within a certain time span leads to unreasonable scheduling of maintenance for parts that will not need repair for a long time. This works well when the detection time is only a few missions long, but when it is longer the consequences outweight the benefits.

One point that should be made here is that, in addition to reducing the variability of A_O by scheduling more regular maintenance visits, Section 5.5.2 found that CBM-MiMOSA observed fewer aborted missions than CBM, even though aborted missions should have equal likelihood of occurring for the two paradigms. This effect increases as the detection lead time grows shorter, leading to an increased benefit from CBM-MiMOSA. The reason this occurs turns out to be interesting and unexpected. At short PHM detection lead times, it turns out that the only difference between the two maintenance paradigms is in the way that they select aircraft to fly missions. The CBM paradigm operates on a strictly first come, first served basis. However, CBM-MiMOSA uses optimization to select aircraft. When very little warning about upcoming part failures is available, as is the case for short PHM detection lead times,

many of the aircraft of the fleet will not predict failures until the mission on which the part will fail. Common sense dictates that neither maintenance paradigm would be able to do anything about this. In the case of CBM only, this is true: these missions all lead to aborts. In the case of CBM-MiMOSA, it is true to a point. *When these aircraft that do not detect failure early enough are selected to fly missions*, the abort occurs no matter what. However, these aircraft are selected to fly missions less often for CBM-MiMOSA than CBM.

This would seem to suggest that some information is available to CBM-MiMOSA that is not available to CBM. However, what is actually occurring is a preference within the optimization for aircraft that are detecting failures. These are preferred within the optimization because one of its three goals is to fly the most missions on aircraft that are predicting failure as these aircraft will soon be repaired, and the optimization is trying to minimize the part life wasted. The effect this has when the PHM detection lead time is short is to place a statistical preference on aircraft that are *less likely to have latent failures*, because the aircraft that have latent failures have no predicted failures and are therefore not flown. This somewhat elegant result represents a suggestion that could be carried into the real world. When true PHM systems have limited time in which failures may be predicted, one option would be to prefer to fly aircraft that are predicting failures soon. However, this does not present a very helpful solution the rest of the time. In this case the decision maker would probably implement a scheduled maintenance paradigm, but hold repairs when signals indicate that the aircraft truly has more time to be flown. Alternately, careful inspection of these parts might be a better option for having reliably performing aircraft.

5.8 Initial Inventory Investment Required

The decision maker's final and most important task is to evaluate the three maintenance paradigm alternatives to determine which is able to best meet a target value of 70% A_O with the least initial inventory investment. However, as Section 5.2 illustrated, A_O should not be explored in isolation; knowing the corresponding value of R_O is important in case the experiment yields unexpected results in this dimension. Table 22 shows A_O and R_O for ten repetitions when the inventory was dialed for each to achieve close to 70% A_O. The inventories required to achieve 70% A_O for each are shown in the last row of Table 22, which shows that across ten repetitions, none of the models were able to exactly match the target A_O level. Though the three models have slightly different A_O results, this is not due to any being inherently "better" than the others. In this case, it is due to the variation between different repetitions, making it hard to exactly match 70% A_O. This is clear from the standard deviation in the A_O average between the ten repetitions.

Table 22: Average across ten repetitions of high level metrics

	Unscheduled	ALIS	Optimization
Average A_O	0.597	0.650	0.676
StDev A_O	0.0631	0.130	0.0760
Min A_O	0.490	0.501	0.539
Max A_O	0.706	0.848	0.780
Average R_O	0.836	0.889	0.832
StDev R_O	0.0418	0.0598	0.0190
Min R_O	0.764	0.7925	0.791
Max R_O	0.912	0.964	0.857
Inventory	75, 10	73, 10	70, 9

In terms of the variability of metrics, CBM-MiMOSA has a higher standard deviation across the average operational availabilities and larger range of average operational availabilities than the reactive maintenance paradigm. It does have better standard deviation and range than CBM, indicating that to some extent the goal of reducing variability from a standard CBM approach succeeded. Due to the fact

that CBM is more likely to be used today than reactive maintenance, being able to reliably reduce the variability in its performance is an important contribution. Given that CBM-MiMOSA is a first attempt at challenging the assumptions underlying CBM, it is likely that another method for rescheduling maintenance events to reduce variability would have even greater success.

Up until now, the metrics for the three maintenance paradigms have been examined across ten reptitions of Sustain-ME, using the aggregate statistics of mean and standard deviation. Another way to examine the behavior of the three paradigms is to look at the variability present *within* a repetition, as the single repetition represents what will be experienced in reality. Figures 115 through 117 shows the operational availability for the reactive maintenance, CBM, and CBM-MiMOSA paradigms respectively. The variability over time is significant at this inventory level for all three paradigms. Even Figure 118, which shows the ten-day rolling average of the operational availability possesses a high degree of variability over time. This indicates that at the operating conditions specified in the Air Force paradigm shift, there is a concerning amount of variability associated with the desired CBM approach, and the modified approach proposed in this thesis does not allow for improvement in this regard. One benefit that was obtained was a slight reduction in inventory at this A_O level, but at the cost of uncertainty it may not be worth the savings.

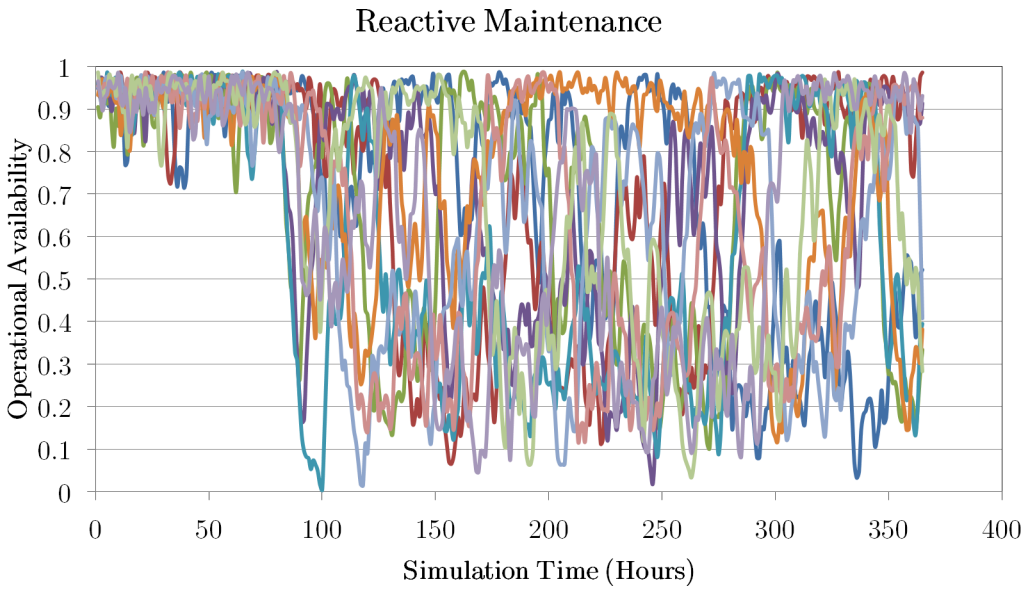


Figure 115: Ten repetitions of operational availability – reactive maintenance

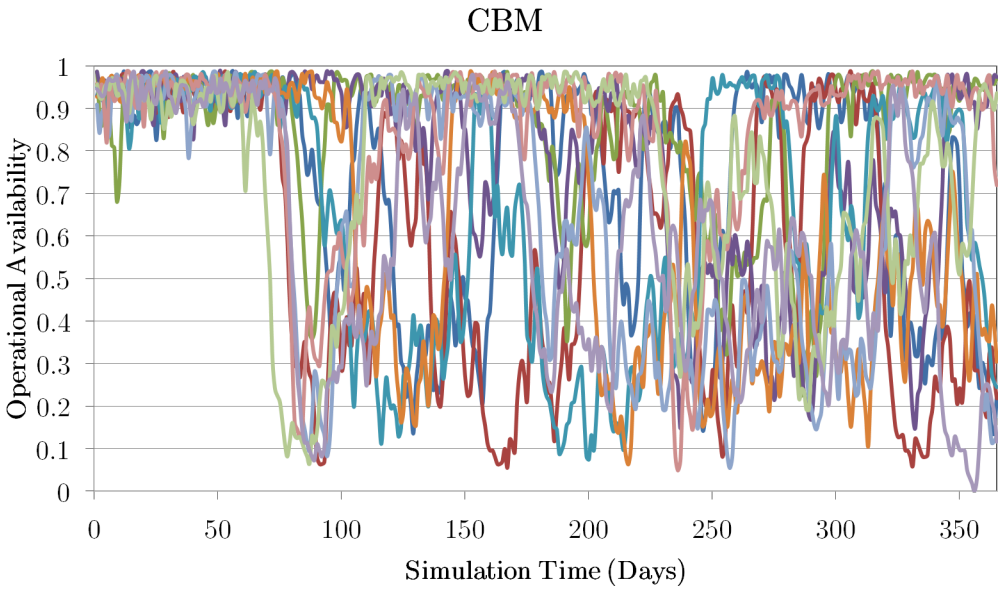


Figure 116: Ten repetitions of operational availability – CBM paradigm

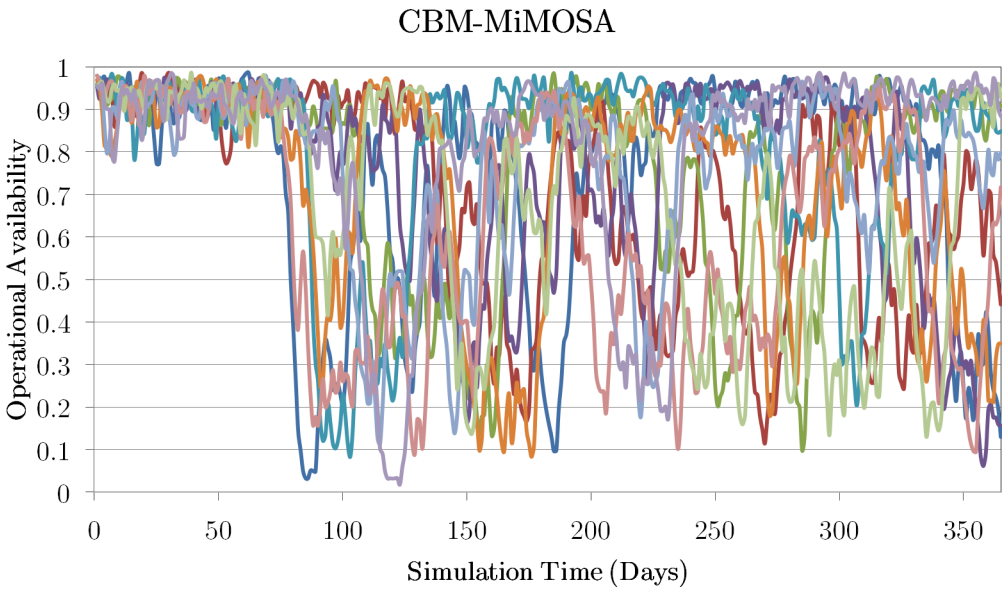


Figure 117: Ten repetitions of operational availability – CBM-MiMOSA

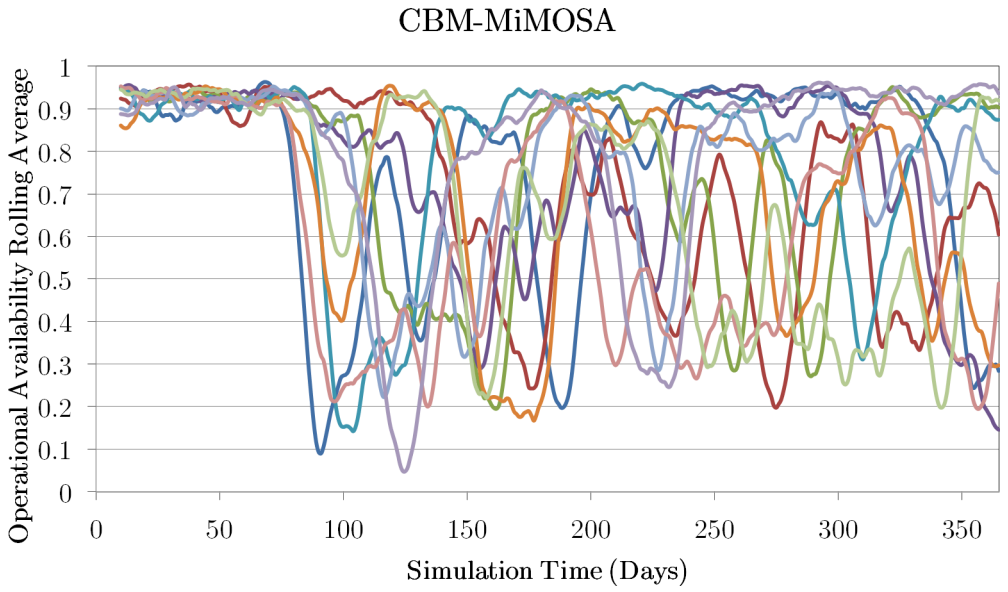


Figure 118: Ten repetitions of operational availability – CBM-MiMOSA

5.9 Use Case Conclusions

The decision maker has now had the opportunity to use Sustain-ME to explore the behavior of some maintenance strategy alternatives. In doing so, she learned a great deal about the potential behavior of sustainment under the Air Force paradigm shift and observed new phenomenon using Sustain-ME that could not have been observed before. First, she observed that the combination of condition based maintenance (under the assumption of just in time maintenance) and minimal inventory is likely to yield nonstationary behavior to an undesirable degree. She observed that one solution, named CBM-MiMOSA, shows promise but has a concerning quality of unduly increasing maintenance events and repairing extremely early under some conditions. Those conditions were determined to be when the PHM detection lead time is earlier than about 90% of part life, but after this region CBM-MiMOSA performed well, in some aspects better even than CBM. CBM, however, was found to be the best strategy among the three tested under the assumptions specific to this study, in that it generally resulted in the highest percentage of missions flown regardless of the value of other parameters. However, CBM-MiMOSA showed promise and it is likely that if another optimization scheme that requires fewer assumptions were to be implemented, it might surpass CBM in a more definitive manner. It is important to remember, though, that all this information is in light of the fact that, at the inventory conditions and A_O level desired, the behavior of sustainment is undesirably variable. This means that beyond any decisions made about how best to implement CBM, some decision is likely to be necessary about how to reduce operational variability. Options that seem clear from the study done in this thesis are to increase inventory and target a higher level of A_O, to use some hybrid of a CBM and scheduled maintenance approach, or to invest significant resources into reducing the variability of maintenance times experienced with the standard CBM paradigm. These are all studies that can be done using Sustain-ME, and the decision maker at this point

would most likely report on the results of the current study, discuss future possible avenues, and do further analysis to refine the sustainment concept going forward.

To close the loop of the use case methodology, Figure 119 shows how the steps of the methodology have been satisfied with different experiments and other work throughout this thesis.

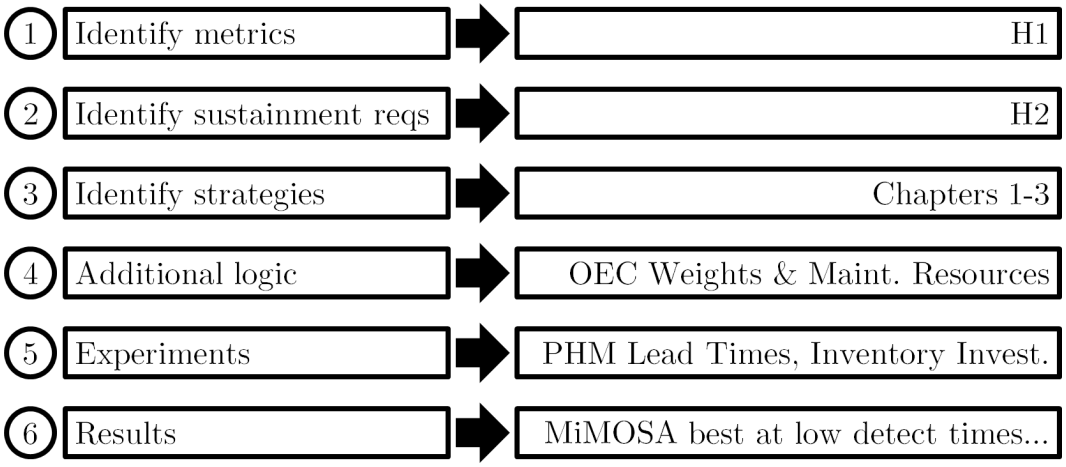


Figure 119: Mapping use case methodology to thesis work

CHAPTER VI

CONCLUSION

This thesis has introduced a problem concerning a conflict of goals for Air Force sustainment contractors. These goals require that exact levels of performance be supported by contractors, but at minimal inventory. Initial examination of the problem suggested this would be difficult; potentially impossible. This thesis developed a modeling environment to determine whether the prediction had any merit, as well as to more broadly be able to study the sustainment problem. Part of this effort was to explain how the environment was developed and tested, as well as to openly acknowledge the assumptions made so that the results can be examined in the proper context. This was done, and Sustain-ME was demonstrated for a sample problem that compared three maintenance paradigms: a baseline reactive maintenance paradigm, a modern condition based maintenance paradigm, and a novel version of the CBM paradigm that adjusts when maintenance visits are scheduled, CBM-MiMOSA. Several aspects of the problem were examined, including the complicated nature of the metrics used to measure sustainment performance, the stochasticity inherent in each of the models at medium inventory levels, and the overall effect of the different maintenance paradigms. Section 6.1 reviews these findings in more depth, and Section 6.2 discusses the contributions made in this thesis as well as future work that could be done and future applications of the modeling environment developed.

6.1 Thesis Results Summary

The first "results" of the thesis are actually the set of verification activities performed in Chapter 4. Though a complete list of these activities will not be shown here, a represenative set are summarized.

1. The generation of sorties was tested by varying the parameters from which the sortie schedule is defined and examining the missions scheduled. Sorties were found to generate as expected.

2. The assignment of sorties was tested by varying the number of aircraft in the fleet. Sorties were found to be assigned as expected.

3. The fleet behavior excluding the supply chain was tested in the following ways:

    3.1. The events carried out by each of the aircraft in the fleet over time were examined and compared against allowable event sequences. The events were found to conform to alloawble event sequences.

    3.2. The frequency of different event sequences was computed and verified.

    3.3. The distributions for different sustainment step durations were compared to histograms of samped values from the simulation and found to show close agreement.

4. The fleet behavior including the supply chain was tested in the following ways:

    4.1. The model with a supply chain was compared at high inventory to the results of the model with an assumed rapid supply chain that was not modeled. The two were found to have similar behavior.

    4.2. The effect of inventory on the operational availability was tested and found to increase the operational availability as expected.

    4.3. The frequency of different aircraft event sequences was again computed and verified.

    4.4. The events carried out by each of the parts in the inventory over time were examined and compared against allowable event sequences. These sequences were found to match the behavior observed.

4.5. The number of parts in each state was tracked over time and the total number was confirmed to remain the same throughout the simulation.

5. The PHM behavior was tested by examining the specific events associated with the PHM and with PHM repair and found to conform to expectations.

6. The coding of CBM-MiMOSA was tested in the following ways:

   6.1. The solutions found by the optimizer were output and examined and found to yield reasonable values.

   6.2. The specific behavior associated with the PHM and CBM-MiMOSA was tracked over time and found to conform to expectations.

   6.3. The event sequences carried out by the fleet's aircraft were again examined and verified.

In addition to the verification activities performed in Chapter 4, Chapter 5 used Sustain-ME to answer a series of questions that were meant as a demonstration of the capabilities of such a modeling environment. Some of these questions were motivated by research questions and hypotheses discussed earlier in the thesis; they are listed here as a reminder.

Research Question 1: Are metrics A_O and R_O both required to capture the behavior of sustainment?

Research Question 1 was to be answered by Hypothesis 1:

Hypothesis 1: The relationship between A_O and R_O is complex and cannot be represented by a simple correlation.

Hypothesis 2: At the conditions cited in the Air Force sustainment paradigm shift, where minimal inventory is selected to meet a target value of 70% A_O, and where maintenance is performed based on a condition based maintenance policy, stochasticity will dominate the performance of sustainment.

Hypothesis 2 was posed by itself as a prediction about the consequences of the Air Force sustainment paradigm shift.

Research Question 3 Does CBM-MiMOSA improve the maintenance visit distribution over time?

Research Question 3 was answered through experiments, but not as a test of a specific hypothesis.

The research questions and hypotheses listed here are the result of disparate aspects of sustainment and maintenance explored throughout the thesis. The answers to these questions and tests of these hypotheses are summarized. Along with these results are summarized the remainder of the results for this thesis, which were derived from the need to demonstrate how a sustainment decision making process can be facilitated by Sustain-ME.

1. A use case methodology was defined for using Sustain-ME to answer sustainment questions.

2. In step 1 of the methodology the metric definition from Chapter 1 was referenced and Hypothesis 1 was tested.

    2.1. Hypothesis 1 was supported by Sustain-ME.

    2.2. Both A_O and R_O were found to be necessary metrics for defining sustainment performance.

3. In step 2 of the methodology the decisions made in creating Sustain-ME were summarized and Hypothesis 2 was tested.

    3.1. Hypothesis 2 was supported by Sustain-ME.

    3.2. High stochasticity was found to occur under the conditions associated with the Air Force sustainment paradigm shift.

4. In step 3 of the methodology the sustainment strategies determined for testing Sustain-ME were summarized.

5. In step 4 of the methodology the weightings associated with CBM-MiMOSA's objective function OEC were tested and the impact of a different maintenance strategy on the maintenance resources required was examined.

    5.1. The weightings were not found to significantly affect the optimization solution.

    5.2. Research Question 3 was posed and answered: CBM-MiMOSA does improve the distribution of maintenance events throughout time.

    5.3. CBM-MiMOSA was found to need more maintenance resources than CBM due to the acceleration in maintenance events due to the optimizer.

6. In step 5 of the methodology the rate of convergence of the mean for experimental data was tested and found to be fairly converged after four, and definitely conveged after ten repetitions.

7. In step 6 of the methodology the results of interest defined in Step 1 of the methodology were analyzed.

    7.1. The effect of the PHM detection lead time was explored for both CBM maintenance paradigms with the same inventory level. Decreasing PHM

detection lead time was found to improve the performance of CBM-MiMOSA but degrade the performance of CBM.

7.2. The ability of each maintenance paradigm to meet a target value of 70% A_O was tested and found to be lacking. However, under the later PHM detection lead time CBM-MiMOSA was able to achieve the same variable performance with less inventory than the other two paradigms.

8. The end result of all these results was that Sustain-ME was found to be an effective way of performing sustainment trade-offs that provided a great deal of information about sustainment decisions to decision makers.

6.2 Contributions and Future Work

The overarching goal of this thesis was to create a modeling environment, Sustain-ME, that allows different potential sustainment decisions to be traded off based on a quantitative modeling environment. Just as important, Sustain-ME was created to be transparent so that future studies can input different assumed values and logics to match other questions about the sustainment process. Because Sustain-ME was created because of the complexities associated with a new Air Force sustainment paradigm shift, it was also demonstrated for maintenance paradigms related to this paradigm shift. In creating Sustain-ME and demonstrating it, several sub-goals were achieved over the course of the thesis and each represents a contribution by itself.

1. A sustainment trade-off modeling environment, Sustain-ME has been developed for studying different sustainment decisions with a common platform.

2. Sustain-ME's behavior has been verified against expected results.

3. Sustain-ME's code has been provided and the process for developing it described so that it can be used and adapted for other problems.

4. Sustain-ME's capabilities have been demonstrated for a use case where three different maintenance paradigms were compared: a reactive maintenance paradigm, a modern CBM paradigm, and CBM-MiMOSA.

5. The optimization problem used to perform CBM-MiMOSA has been mathematically defined.

6. A use case methodology was developed for using Sustain-ME to perform sustainment trade studies such as those that follow.

7. The necessity of examining multipe high level sustainment metrics when describing the performance of sustainment has been demonstrated.

8. The nonstationarity associated with meeting 70% A_O under stochastic maintenance paradigms has been demonstrated.

9. The performance of CBM-MiMOSA has been explored and observations made that would help to further develop this paradigm as a reasonable option for performing maintenance.

10. The effect of PHM detection lead time has been demonstrated for the two CBM-based maintenance paradigms.

11. The problems with achieving exact performance levels at low inventory for a stochastic sustainment process have been shown for all three maintenance paradigms.

The future work associated with this thesis can be split into two categories: the work that arises due to the specific use case used to test Sustain-ME, and the work associated with the model itself. For the former, the primary target for future improvement is CBM-MiMOSA, which showed promise but would require work to be made into a truly viable maintenance option. The main limitation of the method was

based on the specific optimization problem developed to characterize its goals and constraints, and a first step for demonstrating further improvement would be to expand the scope of the problem, potentially with a stochastic optimizer as opposed to a linear program. Future steps might examine other ways to reduce stochasticity of maintenance times without going straight to the option of evenly distributing maintenance events. It would also be interesting to see either version of maintenance optimization compared against one additional baseline: purely scheduled maintenance.

For the latter category of future work related to the use of Sustain-ME in other contexts, the applications are many. Other maintenance paradigms that have been mentioned in this thesis include maintenance-free operating periods, zero maintenance, and phase maintenance, and studies that compare these methods to the three that have been used here would be useful. Outside of a maintenance context, Sustain-ME could be used to test different supply chain policies from the perspective of a vendor who has been contracted to support the sustainment process. As part of this work, a cost component of Sustain-ME could be helpful in determining the relative merits of different options. Finally, Sustain-ME could be used from an operational context to examine different strategies for fielding aircraft on missions. As this thesis demonstrated, there is potential improvement that is not obvious on the surface associated with different aircraft assignment rules. In general this is true of the sustainment process.

6.3 Final Thoughts

This thesis has described the creation, verification and demonstration of a new modeling environment built to facilitate sustainment decision making: Sustain-ME. The paradigm shift that motivated the creation of Sustain-ME has been described, and hypotheses based on this information were stated. Background information about the sustainment process and Sustain-ME development have been provided, the means by

which the hypotheses would be tested were laid out. The reasoning behind a novel maintenance paradigm based on the principles of condition based maintenance, CBM-MiMOSA, was explored; a mathematical function was created to quantify the success CBM-MiMOSA. Sustain-ME was then developed and verified, and it was used to test the hypotheses introduced in the beginning of the thesis. Finally, the way in which Sustain-ME can be used to study different sustainment decisions was demonstrated with a use case based on three maintenance paradigms. These paradigms were studied and found to be valuable under different conditions. Finally, Appendix A will provide the code so that anyone can use Sustain-ME to study sustainment behavior of interest.

APPENDIX A

SUSTAIN-ME CODE

The Sustain-ME modeling environment was coded using Python version 2.7, SimPy version 2.2, and Gurobi version 6.0.0. The code is presented in its entirety below, and annotated to give clarity about the purpose of the different coded modules. These modules are those listed in Figure 22. The complete version of the code with all modules implemented is shown in Figure 120; this figure will be referenced throughout the code.

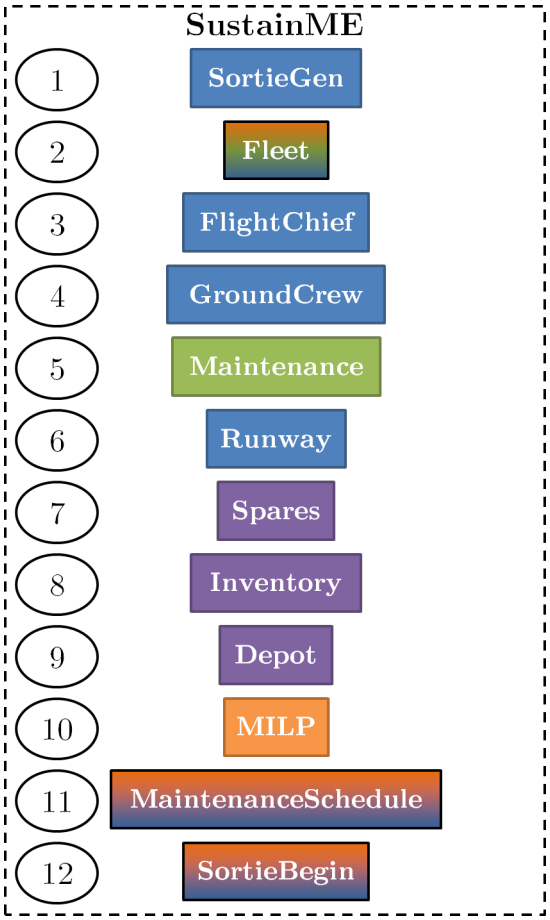


Figure 120: Sustain-ME modules

The lines of code below provide Python with instructions for accessing libraries which will be referenced throughout the remainder of the code. These libraries are freely available and used for doing advanced mathematical computations, generating pseudo-random numbers, processing and outputting data, and performing other complex functions. The most important two libraries used in this thesis were mentioned in the introduction of this appendix, SimPy and gurobi. SimPy is a library that allows discrete event simulations to be built from standard functions, and gurobi is a mixed integer linear program solver.

```
#-------------#
#- Sustain-ME -#
#-------------#

import sys
from SimPy.Simulation import *
import random
import numpy
import math
import csv
from gurobipy import *
```

The lines of code below establish the individual files to which data will be output throughout the simulation. These data files contain different types of verification data that was used throughtout Chapter 4. Some also contain output data that was used to perform the studies in Chapter 5.

```
ac_event_file = open('ACEventVerification.csv', 'w')
part_event_file = open('PartEventVerification.csv', 'w')
phm_event_file = open('PHMEventVerification.csv', 'w')
optimization_file = open('OptimizationInfo.csv', 'w')
mis_prep_file = open('MisPrepInfo.csv', 'w')
ao_file = open('AoVerification.csv', 'w')
ro_file = open('RoVerification.csv', 'w')
all_file = open('all.csv', 'w')
part_file = open('PartVerification.csv', 'w')
test_file = open('testfile.csv', 'w')
fcfile = open('FCVerification.csv', 'w')
gcfile = open('GCVerification.csv', 'w')
ccfile = open('CCVerification.csv', 'w')
```

```
rwfile = open('RWVerification.csv', 'w')
mffile = open('MFVerification.csv', 'w')
msfile = open('MSVerification.csv', 'w')
```

The lines of code below establish the specific decision alternatives that will be run in Sustain-ME. The specific options shown below, PHM=0 or PHM=1 and MaintenanceOptimizer=0 or MaintenanceOptimizer=1 reference the three maintenance paradigms that were modeled in Chapter 5. The reactive maintenance paradigm is run when PHM=0 and MaintenanceOptimizer=0; the traditional CBM paradigm is run when PHM=1 and MaintenanceOptimizer=0; and CBM-MiMOSA is run when PHM=1 and MaintenanceOptimizer=1.

```
#All time steps are in hours

class ModelVersion:
    #The following contains toggles for running different versions
       of the model
    ##PHM determines whether PHM information is used to determine
       when to fix aircraft and order parts
    ##PHM = 0 for no PHM, PHM = 1 for PHM
    PHM = 1
    ##MaintenanceOptimizer = 0 for no optimization, 1 for
       optimization
    MaintenanceOptimizer = 1
```

The lines of code below define the assumptions made in this version of Sustain-ME, justified in Chapter 2 and summarized in Section 4.2. This is where changes should be made to account for different distributions around the duration of the different steps of sustainment, or where different sustainment steps should be defined. This is also where alternate flying and maintenance schedules should be defined, and where different rules for daily aircraft mission limits and rollover missions should be defined.

```
class Assumption:
    #The following designates the time distribution assumptions for
       different model steps
    ##Mission Scheduling
    step_mis_sched_min = .5
```

```
step_mis_sched_mod = .75
step_mis_sched_max = 1.
##Mission Preparation
###Preflight
step_mis_prep_pf_min = .8333
step_mis_prep_pf_mod = 1.
step_mis_prep_pf_max = 1.2
###Refuel
step_mis_prep_rf_min = .3333
step_mis_prep_rf_mod = .3667
step_mis_prep_rf_max = .4167
###Load Weapons
step_mis_prep_lw_min = .75
step_mis_prep_lw_mod = 1.
step_mis_prep_lw_max = 1.25
###Taxi
step_mis_prep_tx_min = .1167
step_mis_prep_tx_mod = .1667
step_mis_prep_tx_max = .2
###Takeoff
step_mis_prep_to_min = .0333
step_mis_prep_to_mod = 0.05
step_mis_prep_to_max = .0667
##Sortie Duration
step_mis_fly_min = 1.3
step_mis_fly_mod = 1.5
step_mis_fly_max = 2.
##Mission Recovery
###Landing
step_mis_rec_land_min = .2333
step_mis_rec_land_mod = .25
step_mis_rec_land_max = .2667
###Parking and Recovery
step_mis_rec_park_min = .0833
step_mis_rec_park_mod = .1167
step_mis_rec_park_max = .15
##Operational Level Repair
###Servicing
step_OL_rep_serv_min = .75
step_OL_rep_serv_mod = 1.
step_OL_rep_serv_max = 1.25
###Document Corrective Actions
step_OL_rep_doc_min = 0.0833
step_OL_rep_doc_mod = 0.1667
step_OL_rep_doc_max = .25
###Local Inventory Wait Time
step_OL_rep_inv_min = 0.5
step_OL_rep_inv_mod = 2.
step_OL_rep_inv_max = 2.5
###Inventory Paperwork
step_OL_rep_pap_min = 0.0833
step_OL_rep_pap_mod = 0.1667
step_OL_rep_pap_max = 0.25
##Shipping
```

```
    ###Depot to base
    step_ship_dep_base_min = 2.4
    step_ship_dep_base_mod = 7.2
    step_ship_dep_base_max = 12.
    ###Base to depot
    step_ship_base_dep_min = 6.
    step_ship_base_dep_max = 12.
    ##Depot Repair
    step_D_rep_min = 1666.56
    step_D_rep_mod = 2083.2
    step_D_rep_max = 2499.84
    #The following designates other assumptions made in this code
    ##Flying hours are 7am to 7pm
    begin_sorties = 7
    end_sorties = 19
    ##Maintenance hours are 24/7
    begin_repair = 0
    end_repair = 24
    repair_sched = [1, 1, 1, 1, 1, 1, 1]
    ##Daily sortie number limit for any given aircraft is 1
    daily_limit = 1
    ##Backlog tracking for 1 week only - unflown missions are rolled
        over but expire after 7 days
    backlog = 7
```

The following lines of code are where several simple model parameters may be changed based on the specific experiment being done with Sustain-ME. The time period over which simulations are run, the operational tempo, the surge profile, and several aircraft definition parameters are contained here. The aircraft parameters are defined for each aggregated part category, so this is where more detailed information could be provided based on the actual aircraft reliability information. In this case, the aircraft is defined by its part reliabilities, the effectiveness of its PHM (relevant only when the PHM=1 option is selected), as well as the distributions associated with the removal and installation of parts. This is also where the fleet size and initial inventory investment are defined.

```
class Input:
    #The following designates the duration of the model and the warm
       -up period
    sim_time = 8760.
    warm_up = 0.
```

```
#The following designates the op tempo (weekly) and surge
    profile (entire simulation)
##Op Tempo: Number of missions to be flown by the fleet each day
     (mission duration is defined in Assumptions)
OT = [10, 10, 10, 10, 10, 10, 10]
##Surge Profile: Multiplier on missions to be flown by the fleet
     over different time intervals
surge_level = [1.]
surge_time = [0., sim_time]

#The following designates the part group inputs - each entry in
    these lists are for a different part group
##Mean Flight Hours Between Repairs: Mean time between events
    requiring a part to resolve
MFHBR = [25., 25., 25., 25., 300., 300.]
lambda_sum = 0
for I in range(len(MFHBR)):
    lambda_sum += 1/MFHBR[I]
aircraft_MFHBR = 1/lambda_sum
#fleet_MTBR = aircraft_MFHBR/(Assumption.step_mis_fly_max*sum(OT
    )/(7*24.))
fleet_MTBR = 9.6
##Removal time: Distribution of time required to remove each
    part group (dist type may change)
###Partially based on literature review
Rem_min = [.75, .75, .75, .75, .75, .75]
Rem_mod = [1., 1., 1., 1., 1., 1.]
Rem_max = [1.25, 1.25, 1.25, 1.25, 1.25, 1.25]
##Install time: Distribution of time required to install each
    part group (dist type may change)
###Partially based on literature review
Ins_min = [1., 1., 1., 1., 1., 1.]
Ins_mod = [1.5, 1.5, 1.5, 1.5, 1.5, 1.5]
Ins_max = [2., 2., 2., 2., 2., 2.]
##Global Repair Turnaround Time: Distribution of time required
    to overhaul the part and return it to 'new' status (dist type
     may change)
##Cost: Refers to part's value, which is relevant to the number
    of parts 'wasted' by repairing before a part has been fully
    used
#cost = [1.]
cost = [1., 12., 1., 12., 1., 12.]
##PHM Detection Lead Time: Distribution of detection time for
    each part as percentage of part life
lead_time_mean = [.80, .80, .80, .80, .80, .80]
lead_time_std = [.05, .05, .05, .05, .05, .05]

#The following designates the fleet variables
##Fleet Size: Number of aircraft in the fleet
num_ac = 30

#The following designates the calibration variables
##Spare Inventory: Number of spares available initially
num_spares = [75, 75, 75, 75, 10, 10]
```

The following lines of code define several output paramters that are collected throughout the model. These are the number of broken parts, the number of parts used, the part life wasted (relevant when the PHM=1 option is selected), and the counts for negative and missed prediction times which collect statistics about the number of truncated PHM detection times.

```
class Output:
    ##None yet - probably need daily as well as overall AO,
       operational reliability, and inventory used
    num_broken = [0]*len(Input.MFHBR)
    num_used = [0]*len(Input.MFHBR)
    part_life_wasted = [0]*len(Input.MFHBR)
    negative_prediction_times = [0]*len(Input.MFHBR)
    missed_prediction_times = [0]*len(Input.MFHBR)
```

The following module is denoted by the number '10' in Figure 120. This module is a function that takes in the current state of the fleet, passed by the variable ACDict, and runs a mixed integer linear program as defined in Chapter 3. This program is then output to tell the Fleet module when to schedule maintenance for the fleet's aircraft and to tell the SortieGen module which aircraft to fly on what day. This is where different logic could be implemented to create another novel maintenance strategy, such as a modification of CBM-MiMOSA to use a stochastic optimizer.

```
def MILP(ACDict):
    #Determine when the last maintenance event was flown
    T_initial = Fleet.last_ME - now()
    if ((now()-7)/24)%7 >=0 and ((now()-7)/24)%7 < 1:
        OT_start = 0
    elif ((now()-7)/24)%7 >=1 and ((now()-7)/24)%7 < 2:
        OT_start = 1
    elif ((now()-7)/24)%7 >=2 and ((now()-7)/24)%7 < 3:
        OT_start = 2
    elif ((now()-7)/24)%7 >=3 and ((now()-7)/24)%7 < 4:
        OT_start = 3
    elif ((now()-7)/24)%7 >=4 and ((now()-7)/24)%7 < 5:
        OT_start = 4
    elif ((now()-7)/24)%7 >=5 and ((now()-7)/24)%7 < 6:
```

```
    OT_start = 5
elif ((now()-7)/24)%7 >=6 or ((now()-7)/24)%7 < 0:
    OT_start = 6
#Sort ACDict and order aircraft by predicting failure soonest to
    latest
ACDict_sorted = sorted(ACDict,key=ACDict.get)
ac_order = [-1]*(len(ACDict)/len(Input.MFHBR))
found = [0]*(len(ACDict)/len(Input.MFHBR))
counter = 0
min_failure_time = {}
for I in range(len(ACDict)/len(Input.MFHBR)):
    while ac_order[I] < 0:
        if found[ACDict_sorted[I+counter][0]] == 0:
            ac_order[I] = ACDict_sorted[I+counter][0]
            found[ACDict_sorted[I+counter][0]] = 1
            min_failure_time[ACDict_sorted[I+counter][0]] =
               ACDict[ACDict_sorted[I+counter][0],ACDict_sorted[
               I+counter][1]]
        else:
            counter += 1
for I in range(len(ac_order)):
    if min_failure_time[I] == 0:
        temp = ac_order.pop(ac_order.index(I))
        ac_order.append(temp)
num_predictingfailure = 0
num_available = 0
for I in range(len(ACDict)/len(Input.MFHBR)):
    if min_failure_time[I] == 100000:
        num_available += 1
    elif min_failure_time[I] > 0:
        num_predictingfailure += 1
aircraft_to_schedule = num_available + num_predictingfailure
time_period = int(math.ceil(Input.fleet_MTBR*
   num_predictingfailure/24.))
RD = {}
for I in range(num_predictingfailure):
    RD[I] = int(math.ceil((I+1)*Input.fleet_MTBR/24))
theoretical_best = 0
for I in range(num_predictingfailure):
    if min_failure_time[I] <> 100000:
        theoretical_best += max(min_failure_time[I]%2,
           min_failure_time[I]-time_period*2)
# Create the model
so = Model("Schedule_Maintenance_Optimization")
so.setParam("OutputFlag",0)
# Create variables
missions = {}
for I in range(aircraft_to_schedule):
    if I+1 <= num_predictingfailure:
        for J in range(RD[I]+1):
            missions[I,J] = so.addVar(vtype=GRB.BINARY,name='m_'
               +str(I)+'_'+str(J))
    else:
        for J in range(max(time_period,1)):
```

```
            missions[I,J] = so.addVar(vtype=GRB.BINARY,name='m_'
                +str(I)+'_'+str(J))
#Maintenance is only performed for aircraft which are predicting
    failure
maintenance_time = {}
for I in range(num_predictingfailure):
    maintenance_time[I] = so.addVar(lb=24*(RD[I]-1)+Assumption.
        begin_sorties+Assumption.step_mis_fly_max+Daily.
        average_PT,ub=24*RD[I]+Assumption.begin_sorties+
        Assumption.step_mis_fly_max+Daily.average_PT,vtype=GRB.
        CONTINUOUS,name='T_' + str(I))
absolute = {}
for I in range(num_predictingfailure):
    absolute[I] = so.addVar(vtype=GRB.CONTINUOUS,name='zeta_'+
        str(I))
obj1 = so.addVar(vtype=GRB.CONTINUOUS,name='obj1')
obj2 = so.addVar(vtype=GRB.CONTINUOUS,name='obj2')
obj3 = so.addVar(vtype=GRB.CONTINUOUS,name='obj3')
# Integrate new variables
so.update()
# Set objective
weights = [1, 1, 1]
so.setObjective(weights[0]*obj1+weights[1]*obj2-weights[2]*obj3,
   GRB.MINIMIZE)
#Add constraint for defining objectives
#Objective 1
            max1 = 0 - T_0 + time_period + (
                num_predictingfailure-1)*MTBR
so.addConstr(obj1*max1-quicksum(absolute[I] for I in range(
   num_predictingfailure))==0,'cobj1')
#Objective 2
            max2 = 0
            for I in range(num_predictingfailure):
                            max2 += min_failure_time[ac_order[I
                                ]]
so.addConstr(obj2*max2-quicksum(min_failure_time[ac_order[I]]-
   Assumption.step_mis_fly_max*quicksum(missions[I,J] for J in
   range(RD[I])) for I in range(num_predictingfailure))==0,'
   cobj2')
#Objective 3
            max3 = OT[0]*time_period
rhs = quicksum(quicksum(missions[I,J] for J in range(RD[I])) for
    I in range(num_predictingfailure))
rhs += quicksum(quicksum(missions[I+num_predictingfailure,J] for
    J in range(max(time_period,1))) for I in range(num_available
   ))
so.addConstr(obj3*max3-rhs==0,'cobj3')
#Add constraint for op tempo
for I in range(max(time_period,1)):
    lhs = 0
    for J in range(num_predictingfailure):
        if RD[J] >= I:
            lhs += missions[J,I]
    for J in range(num_available):
```

```
            lhs += missions[J+num_predictingfailure,I]
        if I == 0:
            so.addConstr(lhs<=sum(SortieGen.sortie_backlog),'c0_'+
                str(I))
        else:
            so.addConstr(lhs<=Input.OT[(OT_start+I)%7],'c0_'+str(I))
    #Add constraint for not violating failure time (repair before
        part actually breaks)
    for I in range(num_predictingfailure):
        so.addConstr(quicksum(Assumption.step_mis_fly_max*missions[I
            ,J] for J in range(RD[I]))<=min_failure_time[ac_order[I
            ]],'c1_'+str(I))
    #Add constraint to enforce absolute value
    if num_predictingfailure > 0:
        so.addConstr(absolute[0]-(maintenance_time[0]-T_initial-
            Input.fleet_MTBR)>=0,'c2.1_0')
        so.addConstr(absolute[0]+(maintenance_time[0]-T_initial-
            Input.fleet_MTBR)>=0,'c2.2_0')
    for I in range(num_predictingfailure-1):
        so.addConstr(absolute[I+1]-(maintenance_time[I+1]-
            maintenance_time[I]-Input.fleet_MTBR)>=0,'c2.1_'+str(I+1)
            )
        so.addConstr(absolute[I+1]+(maintenance_time[I+1]-
            maintenance_time[I]-Input.fleet_MTBR)>=0,'c2.2_'+str(I+1)
            )
    #Add constraint to capture any aircraft which have already flown
        missions today
    for I in range(aircraft_to_schedule):
        if Fleet.AC[ac_order[I]].daily_mission == Assumption.
            daily_limit:
            so.addConstr(missions[I,0]==0,'c3_'+str(I))
    #Add constraint to capture any aircraft which are currently
        unavailable if rerunning the optimization
    if SortieGen.MILP_rerun == 1:
        for I in range(aircraft_to_schedule):
            if Fleet.AC[ac_order[I]] not in Fleet.ACAvailable:
                so.addConstr(missions[I,0]==0,'c4_'+str(I))
    #Add constraint so maintenance times don't keep moving later and
        later
    #Run the optimization
    so.optimize()
    so.write('file.lp')

    #Pull out the necessary variables to return
    total_missions = 0
    schedule = {}
    for I in range(num_predictingfailure):
        for J in range(RD[I]):
            if so.getVarByName('m_'+str(I)+'_'+str(J)).x <> 0:
                total_missions += so.getVarByName('m_'+str(I)+'_'+
                    str(J)).x
    for I in range(num_available):
        for J in range(max(time_period,1)):
```

```
            if so.getVarByName('m_'+str(I+num_predictingfailure)+'_'
               +str(J)).x <> 0:
                total_missions += so.getVarByName('m_'+str(I+
                   num_predictingfailure)+'_'+str(J)).x
    if total_missions > 0:
        for I in range(max(time_period,1)):
            schedule[I] = []
            for J in range(aircraft_to_schedule):
                try:
                    if so.getVarByName('m_'+str(J)+'_'+str(I)).x <>
                       0:
                        schedule[I].append(ac_order[J])
                except:
                    dummy = 0
    else:
        for I in range(max(time_period,1)):
            schedule[I] = []
    repair_times = {}
    for I in range(num_predictingfailure):
        repair_times[ac_order[I]] = so.getVarByName('T_'+str(I)).x +
            now()
    optimization_string = str(now())
    for I in ACDict:
        optimization_string = optimization_string + ',' + str(I) + '
           ,' + str(ACDict[I])
    optimization_string = optimization_string + ',' + str(OT_start)
    for I in schedule:
        optimization_string = optimization_string + ',' + str(I) + '
           ,' + str(schedule[I])
    for I in repair_times:
        optimization_string = optimization_string + ',' + str(I) + '
           ,' + str(repair_times[I])
    for I in min_failure_time:
        optimization_string = optimization_string + ',' + str(I) + '
           ,' + str(min_failure_time[I])
    optimization_file.write(optimization_string + '\n')
    return schedule, repair_times, min_failure_time
```

The following module is denoted by the number '11' in Figure 120. This module was created due to the fact that the MILP module may schedule maintenance for aircraft on a different day than they are scheduled to fly. Under the old logic, if maintenance is scheduled for an aircraft on a day it is not scheduled to fly, the aircraft will never actually be maintained because maintenance is one step in the full aircraft operations loop which is only initiated once a mission has eneded. This module updates that logic and allows for the possibility that an aircraft that is available and waiting for

a mission may have been scheduled for maintenance, and looks for this to occur. If it does, the MaintenanceSchedule module reactivates the aircraft specifically so that it skips the mission steps of the operational loop and proceeds straight to maintenance.

```
class MaintenanceSchedule(Process):
    ID_count = 0
    def __init__(self):
        Process.__init__(self)
        self.ID = MaintenanceSchedule.ID_count
        MaintenanceSchedule.ID_count += 1
        self.day = 0
    def Run(self):
        #Checks for scheduled maintenance times on aircraft which
           are waiting for missions and reactivates them to go
           straight to maintenance
        while 1:
            if self.ID in Fleet.opt_maint.keys() and Fleet.AC[self.
               ID] in Fleet.ACAvailable:
                if Fleet.opt_maint[self.ID]%24 < 7:
                    day = math.floor(Fleet.opt_maint[self.ID]/24.) -
                        1
                else:
                    day = math.ceil(Fleet.opt_maint[self.ID]/24.) -
                       1
                if now() > 24*day + Daily.average_MT and now() <
                   24*(day + 1) + 7:
                    Fleet.AC[self.ID].fly_mission = 0
                    Fleet.AC[self.ID].passivated_for_maintenance = 0
                    Fleet.ACAvailable.pop(Fleet.ACAvailable.index(
                       Fleet.AC[self.ID]))
                    yield hold, self, max(Fleet.opt_maint[self.ID] -
                        now(),0)
                    reactivate(Fleet.AC[self.ID])
                    ac_event_file.write(str(now()) + ',AC ' + str(
                       self.ID) + ',UT,reactivated for maintenance'
                       + '\n')
            elif Fleet.AC[self.ID].hold_for_opt_maint == 1 and Fleet
               .AC[self.ID] in Fleet.ACAvailable:
                Fleet.AC[self.ID].fly_mission = 0
                Fleet.AC[self.ID].passivated_for_maintenance = 0
                Fleet.ACAvailable.pop(Fleet.ACAvailable.index(Fleet.
                   AC[self.ID]))
                yield hold, self, max(Fleet.AC[self.ID].hold_time,0)
                reactivate(Fleet.AC[self.ID])
                ac_event_file.write(str(now()) + ',AC ' + str(self.
                   ID) + ',UT,reactivated for maintenance' + '\n')
            yield hold, self, 0.1
```

The following module is denoted by the number '12' in Figure 120. It implements the

logic that allows the SortieGen module to become inactive until the following day if all aircraft that were scheduled have been assigned to missions.

```
class SortieBegin(Process):
    def __init__(self):
        Process.__init__(self)
    def Run(self):
        while 1:
            if now()%24 <> Assumption.begin_sorties and now() < 24:
                yield hold, self, 1
            if now()%24 == Assumption.begin_sorties:
                reactivate(SortieGen.SortieGen[0])
            elif SortieGen.sortie_aborted > 0 or sum(SortieGen.rerun
               ) > 0:
                reactivate(SortieGen.SortieGen[0])
            yield hold, self, 1
```

The following module is denoted by the number '1' in Figure 120. This is where the logic for how to assign aicraft to missions is defined. The module checks the simulation time and, if it is during the hours that flights may be scheduled, determines the current day's required missions. The unflown missions from the previous n[1] days are rolled over in the backlog as potential missions to fly if aircraft are available. The new day's missions are then placed as the first entry in the backlog. The next step is to look for available aircraft or, if the MILP module has been used, to look at the optimizer's chosen aircraft for the current day. As many missions as there are available aircraft are then assigned and the aircraft reactivated. If the MILP module was used and its full schedule has been assigned, the SortieGen module goes dormant and is reactivated the next day by the SortieBegin module. If not, the aircraft continues to check whether the full backlog has been flown and whether there are aircraft available to fly.

```
class SortieGen(Process):
    SortieGen = []
    sortie_backlog = [0]*Assumption.backlog
    sortie_aborted = 0
    mission_counter = 0
```

[1] n is defined by the backlog from the Assumptions module.

```
MILP_schedule = 0
MILP_day_count = 0
MILP_rerun = 0
MILP_rerun_emptyday = 0
dont_fly = [0]
total_missions_generated = 0
rerun = [0]*Input.num_ac
def __init__(self):
    Process.__init__(self)
    SortieGen.SortieGen.append(self)
    self.passivated = 0
def Run(self):
    #Brings together op tempo, surge profile, and backlog to
       assign available aircraft to sorties
    while 1:
        #Determine how many sorties are to be flown at beginning
            of flight day
        ##If beginning of flight day
        if now()%24 == Assumption.begin_sorties:
            SortieGen.mission_counter = 0
            ##Reset aircraft daily mission counts
            for I in range(Input.num_ac):
                Fleet.AC[I].daily_mission = 0
            SortieGen.sortie_aborted = 0
            ##Erase most previous day of sortie backlog to make
               room for current day's sorties
            SortieGen.sortie_backlog = numpy.roll(SortieGen.
               sortie_backlog,-1)
            ##Check which day of the week it is
            for I in range(len(Input.OT)):
                ##Assign current day's sorties based on the day
                   of the week
                if ((now()-7)/24.)%7 == I:
                    ##As well as the surge profile
                    for J in range(len(Input.surge_level)):
                        if now() >= Input.surge_time[J] and now
                           () < Input.surge_time[J+1]:
                            SortieGen.sortie_backlog[-1] = Input
                               .OT[I]*Input.surge_level[J]
            ##If optimizer has been used, check whether to fly
               missions right now
            if SortieGen.MILP_schedule == 1:
                SortieGen.MILP_day_count += 1
                if SortieGen.MILP_day_count not in Fleet.
                   opt_schedule:
                    SortieGen.min_failure_time = [0]*Input.
                       num_ac
                    for I in range(Input.num_ac):
                        SortieGen.min_failure_time[I] = 100000
                        for J in range(len(Input.MFHBR)):
                            SortieGen.min_failure_time[I] = min(
                               SortieGen.min_failure_time[I],
                               Fleet.ACPHM[(I,J)])
```

```
            ###If any aircraft are available, rerun
               optimization with rerun=1 to see if a new
                schedule can be generated
            if len(Fleet.ACAvailable) > 0 and sum(
               SortieGen.min_failure_time) > 0:
                SortieGen.MILP_rerun = 1
                Fleet.opt_schedule, Fleet.opt_maint,
                   Fleet.opt_part = MILP(Fleet.ACPHM)
                SortieGen.MILP_day_count = 0
                SortieGen.MILP_rerun = 0
                for I in Fleet.opt_schedule[SortieGen.
                   MILP_day_count]:
                    Fleet.AC[I].mission_day = now()/24.
                    Fleet.AC[I].mission_order = 0
                    Fleet.AC[I].mission_order +=
                       SortieGen.mission_counter
                    SortieGen.mission_counter += 1
                    #####Reactivate the selected
                       aircraft so it can go on with its
                        operational cycle
                    reactivate(Fleet.AC[I])
                    #####Delete the selected aircraft
                       from the available aircraft since
                        it can't fly more missions until
                        it has completed the operational
                        cycle
                    Fleet.ACAvailable.pop(Fleet.
                       ACAvailable.index(Fleet.AC[I]))
                    #####Assign the sortie with the
                       highest priority (most recently
                       generated) to the selected
                       aircraft
                    self.sortie_assigned = 0
                    SortieGen.total_missions_generated
                       += 1
                    for I in reversed(range(len(
                       SortieGen.sortie_backlog))):
                        if SortieGen.sortie_backlog[I] >
                            0 and self.sortie_assigned
                           == 0:
                            SortieGen.sortie_backlog[I]
                               -= 1
                            self.sortie_assigned = 1
                self.passivated = 1
                yield passivate, self
            else:
                self.passivated = 1
                yield passivate, self
        elif len(Fleet.opt_schedule[SortieGen.
           MILP_day_count]) == 0 or len(Fleet.
           ACAvailable) == 0:
            ###Check every once in a while for missions
               or new aircraft, but don't fly any now
            self.passivated = 1
```

```
                yield passivate, self
            else:
                ###Check whether all scheduled aircraft are
                    in Fleet.ACAvailable
                SortieGen.MILP_rerun = 0
                for I in Fleet.opt_schedule[SortieGen.
                    MILP_day_count]:
                    if Fleet.AC[I] not in Fleet.ACAvailable:
                        SortieGen.MILP_rerun = 1
                ###If not, rerun optimization with rerun
                    flag set to 1
                if SortieGen.MILP_rerun == 1:
                    Fleet.opt_schedule, Fleet.opt_maint,
                        Fleet.opt_part = MILP(Fleet.ACPHM)
                    SortieGen.MILP_day_count = 0
                    SortieGen.MILP_rerun = 0
                ###Either way, use optimization results to
                    fly all scheduled aircraft
                for I in Fleet.opt_schedule[SortieGen.
                    MILP_day_count]:
                    Fleet.AC[I].mission_day = now()/24.
                    Fleet.AC[I].mission_order = 0
                    Fleet.AC[I].mission_order += SortieGen.
                        mission_counter
                    SortieGen.mission_counter += 1
                    #####Reactivate the selected aircraft so
                        it can go on with its operational
                        cycle
                    reactivate(Fleet.AC[I])
                    #####Delete the selected aircraft from
                        the available aircraft since it can't
                        fly more missions until it has
                        completed the operational cycle
                    Fleet.ACAvailable.pop(Fleet.ACAvailable.
                        index(Fleet.AC[I]))
                    #####Assign the sortie with the highest
                        priority (most recently generated) to
                        the selected aircraft
                    self.sortie_assigned = 0
                    SortieGen.total_missions_generated += 1
                    for I in reversed(range(len(SortieGen.
                        sortie_backlog))):
                        if SortieGen.sortie_backlog[I] > 0
                            and self.sortie_assigned == 0:
                            SortieGen.sortie_backlog[I] -= 1
                            self.sortie_assigned = 1
                self.passivated = 1
                yield passivate, self
    #Option 1: Assign sorties according to optimizer
        schedule
    if SortieGen.MILP_schedule == 1 and now()%24 >=
        Assumption.begin_sorties and now()%24 < Assumption.
        end_sorties:
        if SortieGen.sortie_aborted > 0:
```

```
            SortieGen.MILP_rerun = 1
            Fleet.opt_schedule, Fleet.opt_maint, Fleet.
               opt_part = MILP(Fleet.ACPHM)
            SortieGen.MILP_day_count = 0
            SortieGen.MILP_rerun = 0
            for I in Fleet.opt_schedule[SortieGen.
               MILP_day_count]:
                Fleet.AC[I].mission_day = now()/24.
                Fleet.AC[I].mission_order = 0
                Fleet.AC[I].mission_order += SortieGen.
                   mission_counter
                SortieGen.mission_counter += 1
                #####Reactivate the selected aircraft so it
                   can go on with its operational cycle
                reactivate(Fleet.AC[I])
                #####Delete the selected aircraft from the
                   available aircraft since it can't fly
                   more missions until it has completed the
                   operational cycle
                Fleet.ACAvailable.pop(Fleet.ACAvailable.
                   index(Fleet.AC[I]))
                #####Assign the sortie with the highest
                   priority (most recently generated) to the
                    selected aircraft
                self.sortie_assigned = 0
                SortieGen.total_missions_generated += 1
                for I in reversed(range(len(SortieGen.
                   sortie_backlog))):
                    if SortieGen.sortie_backlog[I] > 0 and
                       self.sortie_assigned == 0:
                        SortieGen.sortie_backlog[I] -= 1
                        self.sortie_assigned = 1
                self.passivated = 1
                yield passivate, self
        elif sum(SortieGen.rerun) > 0:
            for I in range(Input.num_ac):
                if SortieGen.rerun[I] == 1:
                    SortieGen.rerun[I] = 0
            SortieGen.MILP_rerun = 1
            Fleet.opt_schedule, Fleet.opt_maint, Fleet.
               opt_part = MILP(Fleet.ACPHM)
            SortieGen.MILP_day_count = 0
            SortieGen.MILP_rerun = 0
            for I in Fleet.opt_schedule[SortieGen.
               MILP_day_count]:
                Fleet.AC[I].mission_day = now()/24.
                Fleet.AC[I].mission_order = 0
                Fleet.AC[I].mission_order += SortieGen.
                   mission_counter
                SortieGen.mission_counter += 1
                #####Reactivate the selected aircraft so it
                   can go on with its operational cycle
                reactivate(Fleet.AC[I])
```

```
                #####Delete the selected aircraft from the
                    available aircraft since it can't fly
                    more missions until it has completed the
                    operational cycle
                Fleet.ACAvailable.pop(Fleet.ACAvailable.
                    index(Fleet.AC[I]))
                #####Assign the sortie with the highest
                    priority (most recently generated) to the
                     selected aircraft
                self.sortie_assigned = 0
                SortieGen.total_missions_generated += 1
                for I in reversed(range(len(SortieGen.
                    sortie_backlog))):
                    if SortieGen.sortie_backlog[I] > 0 and
                        self.sortie_assigned == 0:
                        SortieGen.sortie_backlog[I] -= 1
                        self.sortie_assigned = 1
                self.passivated = 1
                yield passivate, self
    else:
    #Option 2: Assign sorties according to basic first-come/
        first-served logic
        ##If it is currently flying hours
        if now()%24 >= Assumption.begin_sorties and now()%24
            < Assumption.end_sorties:
            ###While there are missions to fly
            self.first_available = 0
            while sum(SortieGen.sortie_backlog) > 0:
                ####Exit if no aircraft are available
                if len(Fleet.ACAvailable) == 0 or self.
                    first_available >= len(Fleet.ACAvailable)
                    :
                    break
                ####If first available aircraft has not
                    exceeded its mission limits
                if Fleet.ACAvailable[self.first_available].
                    daily_mission < Assumption.daily_limit:
                    #####Tell the selected aircraft what day
                         it was given a mission and which
                        mission was assigned
                    Fleet.ACAvailable[self.first_available].
                        mission_day = now()/24.
                    Fleet.ACAvailable[self.first_available].
                        mission_order = 0
                    Fleet.ACAvailable[self.first_available].
                        mission_order += SortieGen.
                        mission_counter
                    SortieGen.mission_counter += 1
                    #####Reactivate the selected aircraft so
                         it can go on with its operational
                        cycle
                    reactivate(Fleet.ACAvailable[self.
                        first_available])
```

```
                        #####Delete the selected aircraft from
                            the available aircraft since it can't
                             fly more missions until it has
                            completed the operational cycle
                        Fleet.ACAvailable.pop(self.
                            first_available)
                        #####Assign the sortie with the highest
                            priority (most recently generated) to
                             the selected aircraft
                        self.sortie_assigned = 0
                        SortieGen.total_missions_generated += 1
                        for I in reversed(range(len(SortieGen.
                            sortie_backlog))):
                            if SortieGen.sortie_backlog[I] > 0
                                and self.sortie_assigned == 0:
                                SortieGen.sortie_backlog[I] -= 1
                                self.sortie_assigned = 1
                    ####If first available aircraft has exceeded
                        its mission limits
                    else:
                        self.first_available += 1
            if self.passivated == 0:
                yield hold, self, 1
            else:
                self.passivated = 0
```

The following module is denoted by the number '3' in Figure 120. It defines the flight chief resource. This is where the number of flight chiefs should be updated based on the requirements of the study.

```
class FlightChief():
    resource = []
    def __init__(self):
        FlightChief.resource.append(self)
        #There is one flight chief on duty for the fleet at a time
        self.FlightChief = Resource(4)
```

The following module is denoted by the number '4' in Figure 120. It defines the crew chief and ground crew resources. This is where the number of crew chiefs and ground crews should be updated based on the requirements of the study.

```
class GroundCrew():
    resource = []
    def __init__(self):
```

```
        GroundCrew.resource.append(self)
        #There is one crew chief on duty for the fleet at a time
        self.CrewChief = Resource(1)
        #There is one ground crew teams on duty at a time
        self.GroundCrew = Resource(8)
```

The following module is denoted by the number '5' in Figure 120. It defines the maintenance staff and maintenance facility resources. This is where the number of maintenance staff and maintenance facilities should be updated based on the requirements of the study.

```
class Maintenance():
    resource = []
    def __init__(self):
        Maintenance.resource.append(self)
        #There is one maintenance team on duty for the fleet at a
            time
        self.MaintenanceStaff = Resource(1)
        #There is one repair bay available at a time
        self.MaintenanceFacilities = Resource(4)
```

The following module is denoted by the number '6' in Figure 120. It defines the runway resource. This is where the number of runways should be updated based on the requirements of the study.

```
class Runway():
    resource = []
    def __init__(self):
        Runway.resource.append(self)
        #There is one runway available to the fleet
        self.Runway = Resource(2)
```

The following module is denoted by the number '2' in Figure 120. The Fleet module contains all the logic for aircraft operations, including the behavior of the PHM, if the PHM=1 option is selected, and the potential to reactivate and go straight to maintenance, if the MaintenanceOptimizer=1 option is also selected. The full logic

is pictured in Figure 80. This is where many of the decisions made in how to operate the fleet would be updated, depending on the purview of the study. Additional sustainment steps or decisions could be implemented here; also, the PHM's behavior could be updated. The distribution that defines failure for the aircraft's components could also be changed if desired.

```
class Fleet(Process):
    ID_count = 0
    AC = []
    ACAvailable = []
    ACPHM = {}
    for I in range(Input.num_ac):
        for J in range(len(Input.MFHBR)):
            ACPHM[(I,J)] = 100000
    opt_maint = {}
    opt_schedule = {}
    opt_part = {}
    PartQueue = []
    mis_prep_min = 10000
    mis_prep_max = 0
    mis_prep_avg = []
    mis_all_avg = []
    last_ME = 0
    ACPF = 0
    min_failure_time = [0]*Input.num_ac
    maintenance_numbers = []
    maintenance_times = []
    maintenance_aircraft = []
    maintenance_numbers_after = []
    maintenance_parts = {}
    total_missions_flown = 0
    num_detections = [0]*len(Input.MFHBR)
    mission_failures = [0]*len(Input.MFHBR)
    phm_repairs = [0]*len(Input.MFHBR)
    flight_hours = 0
    low_detect = [0]*len(Input.MFHBR)
    one_count = 0
    two_count = 0
    def __init__(self):
        Process.__init__(self)
        self.ID = Fleet.ID_count
        Fleet.ID_count += 1
        Fleet.AC.append(self)
        Fleet.ACAvailable.append(self)
        self.daily_mission = 0
        self.mission_day = -1
        self.mission_order = -1
        self.part_life = [0]*len(Input.MFHBR)
```

```
        self.prediction = [0]*len(Input.MFHBR)
        self.prediction_time = [0]*len(Input.MFHBR)
        self.hold_for_opt_maint = 0
        self.hold_time = 0
        for I in range(len(Input.MFHBR)):
            self.part_life[I] = random.expovariate(1/Input.MFHBR[I])
            #self.prediction_time[I] = self.part_life[I] - random.
               gauss(Input.lead_time_mean[I]*self.part_life[I],Input
               .lead_time_std[I]*self.part_life[I])
            self.prediction_time[I] = random.gauss(Input.
               lead_time_mean[I]*self.part_life[I],Input.
               lead_time_std[I]*self.part_life[I])
            while self.prediction_time[I] < 0 or self.
               prediction_time[I] > self.part_life[I]:
                if self.prediction_time[I] < 0:
                    Output.negative_prediction_times[I] += 1
                elif self.prediction_time[I] > self.part_life[I]:
                    Output.missed_prediction_times[I] += 1
                self.prediction_time[I] = random.gauss(Input.
                   lead_time_mean[I]*self.part_life[I],Input.
                   lead_time_std[I]*self.part_life[I])
            phm_event_file.write(str(now()) + ',AC ' + str(self.ID)
               + ',Part ' + str(I) + ',new part life 1,' + str(self.
               part_life[I]) + ',' + str(self.prediction_time[I]) +
               '\n')
        self.waited = 0
        self.state = 0
        self.t_prev = 0
        self.available_time = 0
        self.flying_time = 0
        self.bar_time = 0
        self.bap_time = 0
        self.bbr_time = 0
        self.passivated_for_mission = 0
        self.fly_mission = 1
        ac_event_file.write(str(now()) + ',AC ' + str(self.ID) + ',
           UT,created' + '\n')
    def Run(self):
        while 1:
            #Wait for a mission (state is available)
            if now()%24 > Assumption.begin_sorties and now()%24 <
               Assumption.end_sorties:
                if self.daily_mission == 0:
                    SortieGen.rerun[self.ID] = 1
            ac_event_file.write(str(now()) + ',AC ' + str(self.ID) +
                ',UT,created' + '\n')
            ac_event_file.write(str(now()) + ',AC ' + str(self.ID) +
                ',UT,awaiting mission' + '\n')
            if self.state == 0:
                self.available_time += now() - self.t_prev
            elif self.state == 1:
                self.flying_time += now() - self.t_prev
            elif self.state == 2:
                self.bar_time += now() - self.t_prev
```

```
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 0
self.t_prev = now()
self.passivated_for_mission = 1
self.fly_mission = 1
yield passivate, self
#No longer passivated for mission
self.passivated_for_mission = 0
if self.fly_mission == 1:
    Fleet.total_missions_flown += 1
    self.t_mis_prep_0 = now()
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,awaiting mission' + '\n')
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,have mission/awaiting f.c.' + '\n')
    ##Seize flight chief and perform mission scheduling
       (state is available)
    fcfile.write(str(now()) + ',' + str(len(FlightChief.
       resource[0].FlightChief.activeQ)) + ',' + str(len
       (FlightChief.resource[0].FlightChief.waitQ)) + ',
       ' + str(len(FlightChief.resource[0].FlightChief.
       activeQ)+len(FlightChief.resource[0].FlightChief.
       waitQ)) + '\n')
    yield request, self, FlightChief.resource[0].
       FlightChief
    fcfile.write(str(now()) + ',' + str(len(FlightChief.
       resource[0].FlightChief.activeQ)) + ',' + str(len
       (FlightChief.resource[0].FlightChief.waitQ)) + ',
       ' + str(len(FlightChief.resource[0].FlightChief.
       activeQ)+len(FlightChief.resource[0].FlightChief.
       waitQ)) + '\n')
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,have mission/awaiting f.c.' + '\n')
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,have f.c./ready to schedule mission' +
        '\n')
    yield hold, self, random.triangular(Assumption.
       step_mis_sched_min, Assumption.step_mis_sched_max
       , Assumption.step_mis_sched_mod)
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,have f.c./ready to schedule mission' +
        '\n')
    ac_event_file.write(str(now()) + ',AC ' + str(self.
       ID) + ',UT,mission scheduled/awaiting g.c.' + '\n
       ')
    ##Release flight chief
```

```
fcfile.write(str(now()) + ',' + str(len(FlightChief.
    resource[0].FlightChief.activeQ)) + ',' + str(len
    (FlightChief.resource[0].FlightChief.waitQ)) + ',
    ' + str(len(FlightChief.resource[0].FlightChief.
    activeQ)+len(FlightChief.resource[0].FlightChief.
    waitQ)) + '\n')
yield release, self, FlightChief.resource[0].
    FlightChief
fcfile.write(str(now()) + ',' + str(len(FlightChief.
    resource[0].FlightChief.activeQ)) + ',' + str(len
    (FlightChief.resource[0].FlightChief.waitQ)) + ',
    ' + str(len(FlightChief.resource[0].FlightChief.
    activeQ)+len(FlightChief.resource[0].FlightChief.
    waitQ)) + '\n')
##Sieze ground crew and perform preflight, refuel,
    and load weapons activities (state is available)
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
yield request, self, GroundCrew.resource[0].
    GroundCrew
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,mission scheduled/awaiting g.c.' + '\n
    ')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have g.c./ready for preflight refuel &
     load weapons' + '\n')
yield hold, self, random.triangular(Assumption.
    step_mis_prep_pf_min, Assumption.
    step_mis_prep_pf_max, Assumption.
    step_mis_prep_pf_mod)
yield hold, self, random.triangular(Assumption.
    step_mis_prep_rf_min, Assumption.
    step_mis_prep_rf_max, Assumption.
    step_mis_prep_rf_mod)
yield hold, self, random.triangular(Assumption.
    step_mis_prep_lw_min, Assumption.
    step_mis_prep_lw_max, Assumption.
    step_mis_prep_lw_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have g.c./ready for preflight refuel &
     load weapons' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,preflight refuel & load weapons done/
    awaiting c.c.' + '\n')
```

```
##Sieze crew chief and perform preflight inspection
    (state is available)
###Currently no time distribution for this step
ccfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].CrewChief.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].CrewChief.waitQ)) + ',' +
    str(len(GroundCrew.resource[0].CrewChief.activeQ)
    +len(GroundCrew.resource[0].CrewChief.waitQ)) + '
    \n')
yield request, self, GroundCrew.resource[0].
    CrewChief
ccfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].CrewChief.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].CrewChief.waitQ)) + ',' +
    str(len(GroundCrew.resource[0].CrewChief.activeQ)
    +len(GroundCrew.resource[0].CrewChief.waitQ)) + '
    \n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,preflight refuel & load weapons done/
    awaiting c.c.' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have c.c./ready for engine start final
     systems check & taxi' + '\n')
ccfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].CrewChief.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].CrewChief.waitQ)) + ',' +
    str(len(GroundCrew.resource[0].CrewChief.activeQ)
    +len(GroundCrew.resource[0].CrewChief.waitQ)) + '
    \n')
yield release, self, GroundCrew.resource[0].
    CrewChief
ccfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].CrewChief.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].CrewChief.waitQ)) + ',' +
    str(len(GroundCrew.resource[0].CrewChief.activeQ)
    +len(GroundCrew.resource[0].CrewChief.waitQ)) + '
    \n')
##Perform engine start, final systems check, and
    taxi activities (state is available)
yield hold, self, random.triangular(Assumption.
    step_mis_prep_tx_min, Assumption.
    step_mis_prep_tx_max, Assumption.
    step_mis_prep_tx_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have c.c./ready for engine start final
     systems check & taxi' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,engine start final systems check &
    taxi done/awaiting r.w.' + '\n')
##Release ground crew, seize runway, and perform
    takeoff (state is flying)
```

```
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
yield release, self, GroundCrew.resource[0].
    GroundCrew
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
###Might need to check if there are rules about how
    long a plane would wait for the runway
rwfile.write(str(now()) + ',' + str(len(Runway.
    resource[0].Runway.activeQ)) + ',' + str(len(
    Runway.resource[0].Runway.waitQ)) + ',' + str(len
    (Runway.resource[0].Runway.activeQ)+len(Runway.
    resource[0].Runway.waitQ)) + '\n')
yield request, self, Runway.resource[0].Runway
rwfile.write(str(now()) + ',' + str(len(Runway.
    resource[0].Runway.activeQ)) + ',' + str(len(
    Runway.resource[0].Runway.waitQ)) + ',' + str(len
    (Runway.resource[0].Runway.activeQ)+len(Runway.
    resource[0].Runway.waitQ)) + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,engine start final systems check &
    taxi done/awaiting r.w.' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have r.w./ready for takeoff' + '\n')
if self.state == 0:
    self.available_time += now() - self.t_prev
elif self.state == 1:
    self.flying_time += now() - self.t_prev
elif self.state == 2:
    self.bar_time += now() - self.t_prev
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 1
self.t_prev = now()
yield hold, self, random.triangular(Assumption.
    step_mis_prep_to_min, Assumption.
    step_mis_prep_to_max, Assumption.
    step_mis_prep_to_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have r.w./ready for takeoff' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,takeoff accomplished/ready for mission
    ' + '\n')
##Release runway (state is flying)
```

```
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
yield release, self, Runway.resource[0].Runway
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
Fleet.mis_prep_min = min(Fleet.mis_prep_min, now() -
    self.t_mis_prep_0)
Fleet.mis_prep_max = max(Fleet.mis_prep_max, now() -
    self.t_mis_prep_0)
Fleet.mis_prep_avg.append(now() - self.t_mis_prep_0)
##Fly mission
self.mis_dur = random.triangular(Assumption.
   step_mis_fly_min, Assumption.step_mis_fly_max,
   Assumption.step_mis_fly_mod)
for I in range(len(Input.MFHBR)):
    phm_event_file.write(str(now()) + ',AC ' + str(
       self.ID) + ',Part ' + str(I) + ',about to fly
       ,' + str(self.part_life[I]) + ',' + str(self.
       prediction_time[I]) + ',' + str(self.mis_dur)
         + ',' + str(Fleet.ACPHM[(self.ID,I)]) + ',\
       n')
yield hold, self, self.mis_dur
Fleet.flight_hours += self.mis_dur
ac_event_file.write(str(now()) + ',AC ' + str(self.
   ID) + ',UT,takeoff accomplished/ready for mission
   ' + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
   ID) + ',UT,mission accomplished/awaiting r.w.' +
   '\n')
##Update ACPHM based on the effect of flying mission
###Only the flying effects should be recorded, not
   the maintenance effects
self.num_r = [0]*len(Input.MFHBR)
self.maintenance = [0]*len(Input.MFHBR)
self.reason = [0]*len(Input.MFHBR)
for I in range(len(Input.MFHBR)):
    if ModelVersion.PHM == 1:
        if self.prediction_time[I] < self.mis_dur
           and self.prediction[I] == 0:
            self.num_detections[I] += 1
            self.prediction[I] = 1
            Fleet.ACPHM[(self.ID,I)] = 0
            Fleet.ACPHM[(self.ID,I)] += self.
               part_life[I]
```

```
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) + ',Part ' + str(I) +
                        ',failure detected,' + str(self.
                       part_life[I]) + ',' + str(self.
                       prediction_time[I]) + ',' + str(self.
                       mis_dur)  + ',' + str(Fleet.ACPHM[(
                       self.ID,I)]) + ',\n')
            if self.part_life[I] < self.mis_dur:
                self.mission_failures[I] += 1
                Fleet.one_count += 1
                self.num_r[I] += 1
                SortieGen.sortie_backlog[-1] += 1
                SortieGen.sortie_aborted += 1
                self.maintenance[I] = 1
                self.reason[I] = 1
                if ModelVersion.PHM == 1:
                    Fleet.ACPHM[(self.ID,I)] = 0
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) +  ',Part ' + str(I)
                       + ',missed a failure,' + str(self.
                       part_life[I]) + ',' + str(self.
                       prediction_time[I]) + ',' + str(self.
                       mis_dur) + ',' + str(Fleet.ACPHM[(
                       self.ID,I)]) + ',\n')
            elif ModelVersion.PHM == 1 and ModelVersion.
               MaintenanceOptimizer == 0 and self.part_life
               < self.mis_dur + Assumption.step_mis_fly_max:
                self.phm_repairs[I] += 1
                self.num_r[I] += 1
                self.maintenance[I] = 1
                self.reason[I] = 2
                Fleet.ACPHM[(self.ID,I)] = 0
                phm_event_file.write(str(now()) + ',AC ' +
                   str(self.ID) + ',Part ' + str(I) + ',
                   detect a failure next mission,' + str(
                   self.part_life[I]) + ',' + str(self.
                   prediction_time[I]) + ',' + str(self.
                   mis_dur)  + ',' + str(Fleet.ACPHM[(self.
                   ID,I)]) + ',\n')
        if sum(self.maintenance) > 0:
            for I in range(len(Input.MFHBR)):
                if ModelVersion.PHM == 1 and self.prediction
                   [I] == 1 and self.num_r[I] == 0 and self.
                   part_life[I] < self.mis_dur + 2*
                   Assumption.step_mis_fly_max:
                    self.num_r[I] += 1
                    self.maintenance[I] = 1
                    self.reason[I] = 2
                    Fleet.ACPHM[(self.ID,I)] = 0
```

```
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) + ',Part ' + str(I) +
                        ',repair due to another repair,' +
                       str(self.part_life[I]) + ',' + str(
                       self.prediction_time[I]) + ',' + str(
                       self.mis_dur)  + ',' + str(Fleet.
                       ACPHM[(self.ID,I)]) + ',\n')
            for I in range(len(Input.MFHBR)):
                if self.reason[I] == 1:
                    Output.num_broken[I] += 1
                    Fleet.two_count += 1
                elif self.reason[I] == 2:
                    Output.part_life_wasted[I] += self.
                       part_life[I] - self.mis_dur
                    self.prediction[I] = 0
                else:
                    self.part_life[I] -= self.mis_dur
                    if ModelVersion.PHM == 1:
                        self.prediction_time[I] = max(0,
                           self.prediction_time[I] - self.
                           mis_dur)
                        if Fleet.ACPHM[(self.ID,I)] > 0 and
                           Fleet.ACPHM[(self.ID,I)] <
                           100000:
                            Fleet.ACPHM[(self.ID,I)] -= self
                                .mis_dur
                            #Fleet.ACPHM[(self.ID,I)] +=
                                self.part_life[I]
                        phm_event_file.write(str(now()) + ',
                           AC ' + str(self.ID) + ',Part ' +
                           str(I) + ',another mission 1,' +
                           str(self.part_life[I]) + ',' +
                           str(self.prediction_time[I]) + ',
                           ' + str(self.mis_dur)  + ',' +
                           str(Fleet.ACPHM[(self.ID,I)]) + '
                           ,\n')
        else:
            for I in range(len(Input.MFHBR)):
                self.part_life[I] -= self.mis_dur
                if ModelVersion.PHM == 1:
                    self.prediction_time[I] = max(0, self.
                       prediction_time[I] - self.mis_dur)
                    if Fleet.ACPHM[(self.ID,I)] > 0 and
                       Fleet.ACPHM[(self.ID,I)] < 100000:
                        Fleet.ACPHM[(self.ID,I)] -= self.
                           mis_dur
                        #Fleet.ACPHM[(self.ID,I)] += self.
                           part_life[I]
```

```
                phm_event_file.write(str(now()) + ',AC '
                    + str(self.ID) + ',Part ' + str(I) +
                    ',another mission 2,' + str(self.
                   part_life[I]) + ',' + str(self.
                   prediction_time[I]) + ',' + str(self.
                   mis_dur)  + ',' + str(Fleet.ACPHM[(
                   self.ID,I)]) + ',\n')
#Seize runway and land (state is flying)
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
yield request, self, Runway.resource[0].Runway
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
   ID) + ',UT,have r.w./ready for landing' + '\n')
yield hold, self, random.triangular(Assumption.
   step_mis_rec_land_min, Assumption.
   step_mis_rec_land_max, Assumption.
   step_mis_rec_land_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
   ID) + ',UT,landing accomplished/awaiting g.c.' +
   '\n')
#Release runway
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
yield release, self, Runway.resource[0].Runway
rwfile.write(str(now()) + ',' + str(len(Runway.
   resource[0].Runway.activeQ)) + ',' + str(len(
   Runway.resource[0].Runway.waitQ)) + ',' + str(len
   (Runway.resource[0].Runway.activeQ)+len(Runway.
   resource[0].Runway.waitQ)) + '\n')
self.daily_mission += 1
#Sieze ground crew and perform park and recovery
   activities (state is available)
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
   resource[0].GroundCrew.activeQ)) + ',' + str(len(
   GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
    str(len(GroundCrew.resource[0].GroundCrew.
   activeQ)+len(GroundCrew.resource[0].GroundCrew.
   waitQ)) + '\n')
yield request, self, GroundCrew.resource[0].
   GroundCrew
```

```
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,have g.c./ready to recover & park' + '
    \n')
if self.state == 0:
    self.available_time += now() - self.t_prev
elif self.state == 1:
    self.flying_time += now() - self.t_prev
elif self.state == 2:
    self.bar_time += now() - self.t_prev
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 0
self.t_prev = now()
yield hold, self, random.triangular(Assumption.
    step_mis_rec_park_min, Assumption.
    step_mis_rec_park_max, Assumption.
    step_mis_rec_park_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',UT,parking & recovery complete/ready for
    servicing' + '\n')
#Perform servicing activities (state is being
    repaired)
if self.state == 0:
    self.available_time += now() - self.t_prev
elif self.state == 1:
    self.flying_time += now() - self.t_prev
elif self.state == 2:
    self.bar_time += now() - self.t_prev
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 4
self.t_prev = now()
yield hold, self, random.triangular(Assumption.
    step_OL_rep_serv_min, Assumption.
    step_OL_rep_serv_max, Assumption.
    step_OL_rep_serv_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',DT,servicing complete/ready for failure
    check' + '\n')
#Release ground crew
```

```
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
yield release, self, GroundCrew.resource[0].
    GroundCrew
gcfile.write(str(now()) + ',' + str(len(GroundCrew.
    resource[0].GroundCrew.activeQ)) + ',' + str(len(
    GroundCrew.resource[0].GroundCrew.waitQ)) + ',' +
     str(len(GroundCrew.resource[0].GroundCrew.
    activeQ)+len(GroundCrew.resource[0].GroundCrew.
    waitQ)) + '\n')
##If the maintenance optimizer should be run, rerun
    it here
if ModelVersion.MaintenanceOptimizer == 1:
    Fleet.min_failure_time = [100000]*Input.num_ac
    for J in range(Input.num_ac):
        for K in range(len(Input.MFHBR)):
            Fleet.min_failure_time[J] = min(Fleet.
                min_failure_time[J],Fleet.ACPHM[(J,K)
                ])
    if sum(Fleet.min_failure_time) > 0:
        optimization_file.write('AC' + str(self.ID)
            + ' ran optimization after flying a
            mission' + '\n')
        Fleet.opt_schedule, Fleet.opt_maint, Fleet.
            opt_part = MILP(Fleet.ACPHM)
        self.min_maint_time = 100000
        for I in Fleet.opt_maint:
            self.min_maint_time = min(self.
                min_maint_time,Fleet.opt_maint[I])
        if self.min_maint_time - Fleet.last_ME >=
            1.5*Input.fleet_MTBR:
            for I in Fleet.opt_maint:
                if Fleet.opt_maint[I] == self.
                    min_maint_time:
                    for J in range(len(Input.MFHBR))
                        :
                        if Fleet.AC[I].part_life[J]
                            == Fleet.opt_part[I]:
                            if Fleet.AC[I].num_r[J]
                                == 0:
                                Output.
                                    part_life_wasted[
                                    J] += self.
                                    part_life[J]
                                Fleet.AC[I].
                                    prediction[J] = 0
                                Fleet.ACPHM[(I,J)] =
                                     0
                                Fleet.AC[I].num_r[J]
                                     += 1
```

```
                                phm_event_file.write(str
                                    (now()) + ',AC ' +
                                    str(I) +  ',Part ' +
                                    str(J) + ',repair due
                                     to exceeding
                                    distance from last ME
                                     parameter 1,' + str(
                                    self.part_life[J]) +
                                    ',' + str(self.
                                    prediction_time[J]) +
                                     ',' + str(self.
                                    mis_dur) + ',' + str(
                                    Fleet.opt_maint[I])
                                    + ',' + str(Fleet.
                                    ACPHM[(I,J)]) + ',\n'
                                    )
                            elif Fleet.AC[I].part_life[J
                                ] < 2*Assumption.
                                step_mis_fly_max and
                                Fleet.AC[I].prediction[J]
                                 == 1:
                                 if Fleet.AC[I].num_r[J]
                                    == 0:
                                     Output.
                                        part_life_wasted[
                                        J] += Fleet.AC[I
                                        ].part_life[J]
                                     Fleet.AC[I].
                                        prediction[J] = 0
                                     Fleet.ACPHM[(I,J)] =
                                         0
                                     Fleet.AC[I].num_r[J]
                                         += 1
                                 phm_event_file.write(str
                                    (now()) + ',AC ' +
                                    str(I) +  ',Part ' +
                                    str(J) + ',repair due
                                     to another repair,'
                                    + str(self.part_life[
                                    J]) + ',' + str(self.
                                    prediction_time[J]) +
                                     ',' + str(self.
                                    mis_dur) + ',' + str(
                                    Fleet.opt_maint[I])
                                    + ',' + str(Fleet.
                                    ACPHM[(I,J)]) + ',\n'
                                    )
                        Fleet.AC[I].hold_for_opt_maint =
                            1
                        Fleet.AC[I].hold_time = self.
                            min_maint_time - now()
        SortieGen.MILP_schedule = 1
        SortieGen.MILP_day_count = 0
```

```
#Compute the maintenance day to determine
   whether the aircraft needs to be flagged for
   maintenance
try:
    if Fleet.opt_maint[self.ID]%24 < 7:
        self.day = math.floor(Fleet.opt_maint[
           self.ID]/24.) - 1
    else:
        self.day = math.ceil(Fleet.opt_maint[
           self.ID]/24.) - 1
except:
    self.day = -100000
if self.ID in Fleet.opt_maint and now() > 24*
   self.day + 7 and now() < 24*(self.day + 1) +
   7:
    for I in range(len(Input.MFHBR)):
        if self.part_life[I] == Fleet.opt_part[
           self.ID]:
            if self.num_r[I] == 0:
                self.num_r[I] += 1
                Output.part_life_wasted[I] +=
                   self.part_life[I]
                self.prediction[I] = 0
                Fleet.ACPHM[(self.ID,I)] = 0
                phm_event_file.write(str(now())
                   + ',AC ' + str(self.ID) +  ',
                   Part ' + str(I) + ',repair
                   due to optimization,' + str(
                   self.part_life[I]) + ',' +
                   str(self.prediction_time[I])
                   + ',' + str(self.mis_dur) + '
                   ,' + str(Fleet.opt_maint[self
                   .ID])  + ',' + str(Fleet.
                   ACPHM[(self.ID,I)]) + ',\n')
        elif self.part_life[I] < 2*Assumption.
           step_mis_fly_max and self.prediction[
           I] == 1:
            if self.num_r[I] == 0:
                self.num_r[I] += 1
                Output.part_life_wasted[I] +=
                   self.part_life[I]
                self.prediction[I] = 0
                Fleet.ACPHM[(self.ID,I)] = 0
                phm_event_file.write(str(now())
                   + ',AC ' + str(self.ID) +  ',
                   Part ' + str(I) + ',repair
                   due to another repair,' + str
                   (self.part_life[I]) + ',' +
                   str(self.prediction_time[I])
                   + ',' + str(self.mis_dur) + '
                   ,' + str(Fleet.opt_maint[self
                   .ID])  + ',' + str(Fleet.
                   ACPHM[(self.ID,I)]) + ',\n')
    self.hold_for_opt_maint = 1
```

```
                self.hold_time = Fleet.opt_maint[self.ID] -
                    now()
        #If an optimization scheduled maintenance event has
            occurred, wait for it and repair
        if self.hold_for_opt_maint == 1:
            yield hold, self, max(self.hold_time,0)
            optimization_file.write(str(now()) + ',' + 'AC'
                + str(self.ID) + ' performed scheduled
                maintenance for part' + str(Fleet.opt_part[
                self.ID]) + '\n')
            self.hold_for_opt_maint = 0
    else:
        ##If doing a maintenance-only run, update parts but
            don't do the mission aspects
        ###If you came from an aircraft that exceeded the
            multiplier*Input.fleet_MTBR limit, do one thing
        if self.hold_for_opt_maint == 1:
            self.hold_for_opt_maint = 0
            for I in range(len(Input.MFHBR)):
                if self.num_r[I] > 0:
                    Output.part_life_wasted[I] += self.
                        part_life[I]
                    self.prediction[I] = 0
                    Fleet.ACPHM[(self.ID,I)] = 0
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) +  ',Part ' + str(I)
                        + ',repair due to optimization,' +
                        str(self.part_life[I]) + ',' + str(
                        self.prediction_time[I]) + ',' + str(
                        Fleet.ACPHM[(self.ID,I)]) + ',\n')
                elif Fleet.ACPHM[(self.ID,I)] < 2*Assumption
                    .step_mis_fly_max and self.prediction[I]
                    == 1:
                    self.num_r[I] += 1
                    Output.part_life_wasted[I] += self.
                        part_life[I]
                    self.prediction[I] = 0
                    Fleet.ACPHM[(self.ID,I)] = 0
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) +  ',Part ' + str(I)
                        + ',repair due to another repair,' +
                        str(self.part_life[I]) + ',' + str(
                        self.prediction_time[I]) + ',' + str(
                        Fleet.ACPHM[(self.ID,I)]) + ',\n')
            optimization_file.write(str(now()) + ',' + 'AC'
                + str(self.ID) + ' performed scheduled
                maintenance for part' + str(Fleet.opt_part[
                self.ID]) + '\n')
        else:
            self.num_r = [0]*len(Input.MFHBR)
            for I in range(len(Input.MFHBR)):
                ###Fix the min failure time part along with
                    any others that will fail on the next
                    mission
```

```
                if self.part_life[I] == Fleet.opt_part[self.
                    ID]:
                    self.num_r[I] += 1
                    Output.part_life_wasted[I] += self.
                        part_life[I]
                    self.prediction[I] = 0
                    Fleet.ACPHM[(self.ID,I)] = 0
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) +  ',Part ' + str(I)
                        + ',repair due to optimization,' +
                        str(self.part_life[I]) + ',' + str(
                        self.prediction_time[I]) + ',' + str(
                        Fleet.ACPHM[(self.ID,I)]) + ',\n')
                elif self.part_life[I] < 2*Assumption.
                    step_mis_fly_max and self.prediction[I]
                    == 1:
                    self.num_r[I] += 1
                    Output.part_life_wasted[I] += self.
                        part_life[I]
                    self.prediction[I] = 0
                    Fleet.ACPHM[(self.ID,I)] = 0
                    phm_event_file.write(str(now()) + ',AC '
                        + str(self.ID) +  ',Part ' + str(I)
                        + ',repair due to another repair,' +
                        str(self.part_life[I]) + ',' + str(
                        self.prediction_time[I]) + ',' + str(
                        Fleet.ACPHM[(self.ID,I)]) + ',\n')
            optimization_file.write(str(now()) + ',' + 'AC'
                + str(self.ID) + ' performed scheduled
                maintenance for part' + str(Fleet.opt_part[
                self.ID]) + '\n')
    Fleet.mis_all_avg.append(now() - self.t_mis_prep_0)
    #Perform failure check
    if sum(self.num_r) > 0:
        Fleet.maintenance_numbers.append(sum(self.num_r))
        Fleet.maintenance_times.append(now())
        Fleet.maintenance_aircraft.append(self.ID)
        Fleet.last_ME = now()
        ac_event_file.write(str(now()) + ',AC ' + str(self.
            ID) + ',DT,failure check complete/awaiting m.f. &
             m.s.' + '\n')
        #Failures have occurred and maintenance must be
            performed
        ##Seize maintenance team and a repair bay (state is
            non mission capable due to repair)
        if self.state == 0:
            self.available_time += now() - self.t_prev
        elif self.state == 1:
            self.flying_time += now() - self.t_prev
        elif self.state == 2:
            self.bar_time += now() - self.t_prev
        elif self.state == 3:
            self.bap_time += now() - self.t_prev
        elif self.state == 4:
```

```
            self.bbr_time += now() - self.t_prev
        self.state = 2
        self.t_prev = now()
        mffile.write(str(now()) + ',' + str(len(Maintenance.
            resource[0].MaintenanceFacilities.activeQ)) + ','
             + str(len(Maintenance.resource[0].
            MaintenanceFacilities.waitQ)) + ',' + str(len(
            Maintenance.resource[0].MaintenanceFacilities.
            activeQ)+len(Maintenance.resource[0].
            MaintenanceFacilities.waitQ)) + '\n')
        yield request, self, Maintenance.resource[0].
            MaintenanceFacilities
        mffile.write(str(now()) + ',' + str(len(Maintenance.
            resource[0].MaintenanceFacilities.activeQ)) + ','
             + str(len(Maintenance.resource[0].
            MaintenanceFacilities.waitQ)) + ',' + str(len(
            Maintenance.resource[0].MaintenanceFacilities.
            activeQ)+len(Maintenance.resource[0].
            MaintenanceFacilities.waitQ)) + '\n')
        msfile.write(str(now()) + ',' + str(len(Maintenance.
            resource[0].MaintenanceStaff.activeQ)) + ',' +
            str(len(Maintenance.resource[0].MaintenanceStaff.
            waitQ)) + ',' + str(len(Maintenance.resource[0].
            MaintenanceStaff.activeQ)+len(Maintenance.
            resource[0].MaintenanceStaff.waitQ)) + '\n')
        yield request, self, Maintenance.resource[0].
            MaintenanceStaff
        msfile.write(str(now()) + ',' + str(len(Maintenance.
            resource[0].MaintenanceStaff.activeQ)) + ',' +
            str(len(Maintenance.resource[0].MaintenanceStaff.
            waitQ)) + ',' + str(len(Maintenance.resource[0].
            MaintenanceStaff.activeQ)+len(Maintenance.
            resource[0].MaintenanceStaff.waitQ)) + '\n')
        ac_event_file.write(str(now()) + ',AC ' + str(self.
            ID) + ',DT,have m.f. & m.s./ready for
            documentation' + '\n')
        ##Document corrective actions (state is non mission
            capable due to repair)
        yield hold, self, random.triangular(Assumption.
            step_OL_rep_doc_min, Assumption.
            step_OL_rep_doc_max, Assumption.
            step_OL_rep_doc_mod)
        ##Proceed with part replacement
        self.parts_filled = [0]*len(Input.MFHBR)
        ac_event_file.write(str(now()) + ',AC ' + str(self.
            ID) + ',DT,documentation complete/awaiting part
            removal' + '\n')
        ##Check whether parts are on hand or not
        self.parts_acquired = [0]*len(Input.MFHBR)
        for I in range(len(Input.MFHBR)):
            if len(Spares.local_inventory_Dict[(I,)]) >=
                self.num_r[I]:
                ###Change part states from local inventory
                    to in transit
```

```
                for J in range(self.num_r[I]):
                    Spares.local_transit_Dict[(I,)].append(
                       Spares.local_inventory_Dict[(I,)].pop
                       ())
                    part_event_file.write(str(now()) + ',
                       Part ' + str(Spares.
                       local_transit_Dict[(I,)][-1].ID) + ',
                       sending part to maintenance facility'
                        + '\n')
                    for K in range(len(Input.MFHBR)):
                        part_string = str(now()/24.) + ',
                           Part category' + str(K+1)
                        temp = 0
                        for L in range(Input.num_ac):
                            temp += len(Spares.equipped_Dict
                               [(K,L)])
                        temp += len(Spares.
                           local_transit_Dict[(K,)])
                        temp += len(Spares.
                           shipping_to_depot_Dict[(K,)])
                        temp += len(Spares.
                           shipping_to_base_Dict[(K,)])
                        temp2 = temp + len(Spares.
                           local_inventory_Dict[(K,)]) + len
                           (Spares.depot_repair_Dict[(K,)])
                        part_string = part_string + ',' +
                           str(len(Spares.
                           local_inventory_Dict[(K,)])) + ',
                           ' + str(len(Spares.
                           depot_repair_Dict[(K,)])) + ',' +
                            str(temp) + ',' + str(temp2) + '
                           \n'
                        part_file.write(part_string)
                    self.parts_filled[I] += 1
                self.parts_acquired[I] = 1
            elif len(Spares.local_inventory_Dict[(I,)]) > 0:
                for J in range(len(Spares.
                   local_inventory_Dict[(I,)])):
                    Spares.local_transit_Dict[(I,)].append(
                       Spares.local_inventory_Dict[(I,)].pop
                       ())
                    part_event_file.write(str(now()) + ',
                       Part ' + str(Spares.
                       local_transit_Dict[(I,)][-1].ID) + ',
                       sending part to maintenance facility'
                        + '\n')
                    for K in range(len(Input.MFHBR)):
                        part_string = str(now()/24.) + ',
                           Part category' + str(K+1)
                        temp = 0
                        for L in range(Input.num_ac):
                            temp += len(Spares.equipped_Dict
                               [(K,L)])
```

```
                    temp += len(Spares.
                       local_transit_Dict[(K,)])
                    temp += len(Spares.
                       shipping_to_depot_Dict[(K,)])
                    temp += len(Spares.
                       shipping_to_base_Dict[(K,)])
                    temp2 = temp + len(Spares.
                       local_inventory_Dict[(K,)]) + len
                       (Spares.depot_repair_Dict[(K,)])
                    part_string = part_string + ',' +
                       str(len(Spares.
                       local_inventory_Dict[(K,)])) + ',
                       ' + str(len(Spares.
                       depot_repair_Dict[(K,)])) + ',' +
                        str(temp) + ',' + str(temp2) + '
                       \n'
                    part_file.write(part_string)
                self.parts_filled[I] += 1
##Remove all relevant LRU from aircraft in parallel
   (state is being repaired)
if self.state == 0:
    self.available_time += now() - self.t_prev
elif self.state == 1:
    self.flying_time += now() - self.t_prev
elif self.state == 2:
    self.bar_time += now() - self.t_prev
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 4
self.t_prev = now()
self.max_removal_time = 0
for I in range(len(Input.MFHBR)):
    if self.num_r[I] > 0:
        self.max_removal_time = max(self.
           max_removal_time, random.triangular(Input
           .Rem_min[I], Input.Rem_max[I], Input.
           Rem_mod[I]))
yield hold, self, self.max_removal_time
##Send broken parts away
for I in range(len(Input.MFHBR)):
    for J in range(self.num_r[I]):
        De = Depot(I, self.ID)
        activate(De,De.Run())
ac_event_file.write(str(now()) + ',AC ' + str(self.
   ID) + ',DT,part removal complete/awaiting new
   part' + '\n')
##If all parts were not acquired before, try to
   acquire them now
self.waited = 0
if sum(self.parts_acquired) < len(Input.MFHBR):
    ###Check whether part is on hand or not
    for I in range(len(Input.MFHBR)):
```

```
                if self.parts_acquired[I] == 0:
                    if len(Spares.local_inventory_Dict[(I,)
                       ]) >= self.num_r[I] - self.
                       parts_filled[I]:
                        ####Change part state from local
                           inventory to in transit
                        for J in range(self.num_r[I] - self.
                           parts_filled[I]):
                            Spares.local_transit_Dict[(I,)].
                               append(Spares.
                               local_inventory_Dict[(I,)].
                               pop())
                            part_event_file.write(str(now())
                                + ',Part ' + str(Spares.
                               local_transit_Dict[(I,)][-1].
                               ID) + ',sending part to
                               maintenance facility' + '\n')
                            for K in range(len(Input.MFHBR))
                               :
                                part_string = str(now()/24.)
                                    + ',Part category' + str
                                   (K+1)
                                temp = 0
                                for L in range(Input.num_ac)
                                   :
                                    temp += len(Spares.
                                       equipped_Dict[(K,L)])
                                temp += len(Spares.
                                   local_transit_Dict[(K,)])
                                temp += len(Spares.
                                   shipping_to_depot_Dict[(K
                                   ,)])
                                temp += len(Spares.
                                   shipping_to_base_Dict[(K
                                   ,)])
                                temp2 = temp + len(Spares.
                                   local_inventory_Dict[(K,)
                                   ]) + len(Spares.
                                   depot_repair_Dict[(K,)])
                                part_string = part_string +
                                   ',' + str(len(Spares.
                                   local_inventory_Dict[(K,)
                                   ])) + ',' + str(len(
                                   Spares.depot_repair_Dict
                                   [(K,)])) + ',' + str(temp
                                   ) + ',' + str(temp2) + '\
                                   n'
                                part_file.write(part_string)
                            self.parts_filled[I] += 1
                        self.parts_acquired[I] = 1
                    elif len(Spares.local_inventory_Dict[(I
                       ,)]) > 0:
                        ####Change part state from local
                           inventory to in transit
```

```
                    for J in range(len(Spares.
                       local_inventory_Dict[(I,)])):
                        Spares.local_transit_Dict[(I,)].
                           append(Spares.
                           local_inventory_Dict[(I,)].
                           pop())
                        part_event_file.write(str(now())
                            + ',Part ' + str(Spares.
                           local_transit_Dict[(I,)][-1].
                           ID) + ',sending part to
                           maintenance facility' + '\n')
                        for K in range(len(Input.MFHBR))
                           :
                            part_string = str(now()/24.)
                                + ',Part category' + str
                               (K+1)
                            temp = 0
                            for L in range(Input.num_ac)
                               :
                                temp += len(Spares.
                                   equipped_Dict[(K,L)])
                            temp += len(Spares.
                               local_transit_Dict[(K,)])
                            temp += len(Spares.
                               shipping_to_depot_Dict[(K
                               ,)])
                            temp += len(Spares.
                               shipping_to_base_Dict[(K
                               ,)])
                            temp2 = temp + len(Spares.
                               local_inventory_Dict[(K,)
                               ]) + len(Spares.
                               depot_repair_Dict[(K,)])
                            part_string = part_string +
                               ',' + str(len(Spares.
                               local_inventory_Dict[(K,)
                               ])) + ',' + str(len(
                               Spares.depot_repair_Dict
                               [(K,)])) + ',' + str(temp
                               ) + ',' + str(temp2) + '\
                               n'
                            part_file.write(part_string)
                        self.parts_filled[I] += 1
                    self.waited = 1
                else:
                    ####Indicate that the parts queue
                       must be joined
                    self.waited = 1
    ###If it is necessary to wait for parts, join
       the parts queue
    if self.waited == 1:
        ac_event_file.write(str(now()) + ',AC ' +
           str(self.ID) + ',DT,still awaiting new
           part' + '\n')
```

```
            ####Continue to wait for parts to be
               available, but release maintenance staff
               (state is non mission capable due to
               supply)
            if self.state == 0:
                self.available_time += now() - self.
                   t_prev
            elif self.state == 1:
                self.flying_time += now() - self.t_prev
            elif self.state == 2:
                self.bar_time += now() - self.t_prev
            elif self.state == 3:
                self.bap_time += now() - self.t_prev
            elif self.state == 4:
                self.bbr_time += now() - self.t_prev
            self.state = 3
            self.t_prev = now()
            msfile.write(str(now()) + ',' + str(len(
               Maintenance.resource[0].MaintenanceStaff.
               activeQ)) + ',' + str(len(Maintenance.
               resource[0].MaintenanceStaff.waitQ)) + ',
               ' + str(len(Maintenance.resource[0].
               MaintenanceStaff.activeQ)+len(Maintenance
               .resource[0].MaintenanceStaff.waitQ)) + '
               \n')
            yield release, self, Maintenance.resource
               [0].MaintenanceStaff
            msfile.write(str(now()) + ',' + str(len(
               Maintenance.resource[0].MaintenanceStaff.
               activeQ)) + ',' + str(len(Maintenance.
               resource[0].MaintenanceStaff.waitQ)) + ',
               ' + str(len(Maintenance.resource[0].
               MaintenanceStaff.activeQ)+len(Maintenance
               .resource[0].MaintenanceStaff.waitQ)) + '
               \n')
            ####Call something to check periodically for
                parts
            Fleet.PartQueue.append(self)
            yield passivate, self
            ac_event_file.write(str(now()) + ',AC ' +
               str(self.ID) + ',DT,have new part/
               awaiting part transfer' + '\n')
    ##Wait for parts to be sent (state is non mission
       capable due to supply) and, if necessary, request
        maintenance staff once more (staff is non
       mission capable due to repair)
    yield hold, self, random.triangular(Assumption.
       step_OL_rep_inv_min, Assumption.
       step_OL_rep_inv_max, Assumption.
       step_OL_rep_inv_mod)
    if self.waited == 1:
        ac_event_file.write(str(now()) + ',AC ' + str(
           self.ID) + ',DT,part transferred/awaiting m.s
           .' + '\n')
```

```
        if self.state == 0:
            self.available_time += now() - self.t_prev
        elif self.state == 1:
            self.flying_time += now() - self.t_prev
        elif self.state == 2:
            self.bar_time += now() - self.t_prev
        elif self.state == 3:
            self.bap_time += now() - self.t_prev
        elif self.state == 4:
            self.bbr_time += now() - self.t_prev
        self.state = 2
        self.t_prev = now()
        msfile.write(str(now()) + ',' + str(len(
           Maintenance.resource[0].MaintenanceStaff.
           activeQ)) + ',' + str(len(Maintenance.
           resource[0].MaintenanceStaff.waitQ)) + ',' +
           str(len(Maintenance.resource[0].
           MaintenanceStaff.activeQ)+len(Maintenance.
           resource[0].MaintenanceStaff.waitQ)) + '\n')
        yield request, self, Maintenance.resource[0].
           MaintenanceStaff
        msfile.write(str(now()) + ',' + str(len(
           Maintenance.resource[0].MaintenanceStaff.
           activeQ)) + ',' + str(len(Maintenance.
           resource[0].MaintenanceStaff.waitQ)) + ',' +
           str(len(Maintenance.resource[0].
           MaintenanceStaff.activeQ)+len(Maintenance.
           resource[0].MaintenanceStaff.waitQ)) + '\n')
        ac_event_file.write(str(now()) + ',AC ' + str(
           self.ID) + ',DT,have m.s./awaiting part
           installation' + '\n')
    else:
        ac_event_file.write(str(now()) + ',AC ' + str(
           self.ID) + ',DT,part transferred/awaiting
           part installation' + '\n')
    ##Once parts and maintenance staff are on hand
       proceed with repairs (state is being repaired)
    if self.state == 0:
        self.available_time += now() - self.t_prev
    elif self.state == 1:
        self.flying_time += now() - self.t_prev
    elif self.state == 2:
        self.bar_time += now() - self.t_prev
    elif self.state == 3:
        self.bap_time += now() - self.t_prev
    elif self.state == 4:
        self.bbr_time += now() - self.t_prev
    self.state = 4
    self.t_prev = now()
    self.max_install_time = 0
    for I in range(len(Input.MFHBR)):
        if self.num_r[I] > 0:
```

```
            self.max_install_time = max(self.
                max_install_time, random.triangular(Input
                .Ins_min[I], Input.Ins_max[I], Input.
                Ins_mod[I]))
        yield hold, self, self.max_install_time
        ##Change part states from in transit to equipped,
            update part life, and prediction time and
            prediction (if relevant) for all parts that were
            replaced
        for I in range(len(Input.MFHBR)):
            for J in range(self.num_r[I]):
                Spares.equipped_Dict[(I,self.ID)].append(
                    Spares.local_transit_Dict[(I,)].pop())
                Output.num_used[I] += 1
                part_event_file.write(str(now()) + ',Part '
                    + str(Spares.equipped_Dict[(I,self.ID)
                    ][-1].ID) + ',reinstalling part on
                    aircraft' + '\n')
                for K in range(len(Input.MFHBR)):
                    part_string = str(now()/24.) + ',Part
                        category' + str(K+1)
                    temp = 0
                    for L in range(Input.num_ac):
                        temp += len(Spares.equipped_Dict[(K,
                            L)])
                    temp += len(Spares.local_transit_Dict[(K
                        ,)])
                    temp += len(Spares.
                        shipping_to_depot_Dict[(K,)])
                    temp += len(Spares.shipping_to_base_Dict
                        [(K,)])
                    temp2 = temp + len(Spares.
                        local_inventory_Dict[(K,)]) + len(
                        Spares.depot_repair_Dict[(K,)])
                    part_string = part_string + ',' + str(
                        len(Spares.local_inventory_Dict[(K,)
                        ])) + ',' + str(len(Spares.
                        depot_repair_Dict[(K,)])) + ',' + str
                        (temp) + ',' + str(temp2) + '\n'
                    part_file.write(part_string)
                self.part_life[I] = random.expovariate(1/
                    Input.MFHBR[I])
                if ModelVersion.PHM == 1:
                    self.prediction_time[I] = random.gauss(
                        Input.lead_time_mean[I]*self.
                        part_life[I],Input.lead_time_std[I]*
                        self.part_life[I])
                    while self.prediction_time[I] < 0 or
                        self.prediction_time[I] > self.
                        part_life[I]:
                        if self.prediction_time[I] < 0:
                            Output.negative_prediction_times
                                [I] += 1
```

```
                elif self.prediction_time[I] > self.
                    part_life[I]:
                     Output.missed_prediction_times[I
                         ] += 1
                self.prediction_time[I] = random.
                    gauss(Input.lead_time_mean[I]*
                    self.part_life[I],Input.
                    lead_time_std[I]*self.part_life[I
                    ])
            self.prediction[I] = 0
            phm_event_file.write(str(now()) + ',AC '
                + str(self.ID) + ',Part ' + str(I) +
                ',new part life 2,' + str(self.
               part_life[I]) + ',' + str(self.
               prediction_time[I]) + ',' + str(self.
               mis_dur)  + ',' + str(Fleet.ACPHM[(
               self.ID,I)]) + ',\n')
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',DT,part installed/ready for paperwork' +
    '\n')
##Document corrective actions (state is non mission
    capable due to repair)
if self.state == 0:
    self.available_time += now() - self.t_prev
elif self.state == 1:
    self.flying_time += now() - self.t_prev
elif self.state == 2:
    self.bar_time += now() - self.t_prev
elif self.state == 3:
    self.bap_time += now() - self.t_prev
elif self.state == 4:
    self.bbr_time += now() - self.t_prev
self.state = 2
self.t_prev = now()
yield hold, self, random.triangular(Assumption.
    step_OL_rep_pap_min, Assumption.
    step_OL_rep_pap_max, Assumption.
    step_OL_rep_pap_mod)
ac_event_file.write(str(now()) + ',AC ' + str(self.
    ID) + ',DT,paperwork done' + '\n')
##Release maintenance team and repair bay
mffile.write(str(now()) + ',' + str(len(Maintenance.
    resource[0].MaintenanceFacilities.activeQ)) + ','
     + str(len(Maintenance.resource[0].
    MaintenanceFacilities.waitQ)) + ',' + str(len(
    Maintenance.resource[0].MaintenanceFacilities.
    activeQ)+len(Maintenance.resource[0].
    MaintenanceFacilities.waitQ)) + '\n')
yield release, self, Maintenance.resource[0].
    MaintenanceFacilities
```

```
mffile.write(str(now()) + ',' + str(len(Maintenance.
   resource[0].MaintenanceFacilities.activeQ)) + ','
    + str(len(Maintenance.resource[0].
   MaintenanceFacilities.waitQ)) + ',' + str(len(
   Maintenance.resource[0].MaintenanceFacilities.
   activeQ)+len(Maintenance.resource[0].
   MaintenanceFacilities.waitQ)) + '\n')
msfile.write(str(now()) + ',' + str(len(Maintenance.
   resource[0].MaintenanceStaff.activeQ)) + ',' +
   str(len(Maintenance.resource[0].MaintenanceStaff.
   waitQ)) + ',' + str(len(Maintenance.resource[0].
   MaintenanceStaff.activeQ)+len(Maintenance.
   resource[0].MaintenanceStaff.waitQ)) + '\n')
yield release, self, Maintenance.resource[0].
   MaintenanceStaff
msfile.write(str(now()) + ',' + str(len(Maintenance.
   resource[0].MaintenanceStaff.activeQ)) + ',' +
   str(len(Maintenance.resource[0].MaintenanceStaff.
   waitQ)) + ',' + str(len(Maintenance.resource[0].
   MaintenanceStaff.activeQ)+len(Maintenance.
   resource[0].MaintenanceStaff.waitQ)) + '\n')
for I in range(len(Input.MFHBR)):
    if self.num_r[I] > 0 and ModelVersion.PHM == 1:
        Fleet.ACPHM[(self.ID,I)] = 100000
        phm_event_file.write(str(now()) + ',AC ' +
           str(self.ID) + ',Part ' + str(I) + ',
           fully reset part,' + str(self.part_life[I
           ]) + ',' + str(self.prediction_time[I])
           + ',' + str(Fleet.ACPHM[(self.ID,I)]) + '
           ,\n')
self.num_r = [0]*len(Input.MFHBR)
optimization_file.write('AC' + str(self.ID) + ' ran
   optimization after flying a mission' + '\n')
if ModelVersion.MaintenanceOptimizer == 1:
    Fleet.opt_schedule, Fleet.opt_maint, Fleet.
       opt_part = MILP(Fleet.ACPHM)
    self.min_maint_time = 100000
    for I in Fleet.opt_maint:
        self.min_maint_time = min(self.
           min_maint_time,Fleet.opt_maint[I])
    if self.min_maint_time - Fleet.last_ME >= 1.5*
       Input.fleet_MTBR:
        for I in Fleet.opt_maint:
            if Fleet.opt_maint[I] == self.
               min_maint_time:
                for J in range(len(Input.MFHBR)):
                    if Fleet.AC[I].part_life[J] ==
                       Fleet.opt_part[I]:
                        Output.part_life_wasted[J]
                           += self.part_life[J]
                        Fleet.AC[I].prediction[J] =
                           0
                        Fleet.ACPHM[(I,J)] = 0
                        Fleet.AC[I].num_r[J] += 1
```

```
                                    phm_event_file.write(str(now
                                        ()) + ',AC ' + str(I) +
                                        ',Part ' + str(J) + ',
                                        repair due to exceeding
                                        distance from last ME
                                        parameter 2,' + str(self.
                                        part_life[J]) + ',' + str
                                        (self.prediction_time[J])
                                         + ',' + str(self.mis_dur
                                        ) + ',' + str(Fleet.
                                        opt_maint[I])  + ',' +
                                        str(Fleet.ACPHM[(I,J)]) +
                                         ',\n')
                                elif Fleet.AC[I].part_life[J] <
                                    2*Assumption.step_mis_fly_max
                                     and Fleet.AC[I].prediction[J
                                    ] == 1:
                                     Output.part_life_wasted[J]
                                        += Fleet.AC[I].part_life[
                                        J]
                                     Fleet.AC[I].prediction[J] =
                                        0
                                     Fleet.ACPHM[(I,J)] = 0
                                     Fleet.AC[I].num_r[J] += 1
                                     phm_event_file.write(str(now
                                        ()) + ',AC ' + str(I) +
                                        ',Part ' + str(J) + ',
                                        repair due to another
                                        repair,' + str(self.
                                        part_life[J]) + ',' + str
                                        (self.prediction_time[J])
                                         + ',' + str(self.mis_dur
                                        ) + ',' + str(Fleet.
                                        opt_maint[I])  + ',' +
                                        str(Fleet.ACPHM[(I,J)]) +
                                         ',\n')
                            Fleet.AC[I].hold_for_opt_maint = 1
                            Fleet.AC[I].hold_time = self.
                                min_maint_time - now()
                SortieGen.MILP_schedule = 1
                SortieGen.MILP_day_count = 0
            Fleet.maintenance_numbers_after.append(1)
        Fleet.ACAvailable.append(self)
        ac_event_file.write(str(now()) + ',AC ' + str(self.ID) +
            ',UT,awaiting mission' + '\n')
```

The following module has been denoted by the number '7' in Figure 120. It initializes the states of the parts in the simulation as either equipped on the aircraft, or in local inventory. It also defines the full set of states that parts may occupy, and

these states must be mutually exclusive and all-encompassing. These dictionaries are containers for part objects, which are moved from container to container as the parts are moved throughout the supply chain cycle. Through this setup, part objects can be constantly tracked to ensure that a continuous transfer throughout the simulation occurs. This is where any additional part logic should be implemented, for instance if the user wanted to create different levels of inventories at different delay times from which the fleet could order. One implementation of this would be to model not only refurbishment of parts, but also manufacturing when necessary.

```
class Spares():
    ID_count = 0
    equipped_Dict = {}
    local_transit_Dict = {}
    local_inventory_Dict = {}
    shipping_to_depot_Dict = {}
    depot_repair_Dict = {}
    shipping_to_base_Dict = {}
    for I in range(len(Input.MFHBR)):
        for J in range(Input.num_ac):
            equipped_Dict[(I,J)] = []
        local_transit_Dict[(I,)] = []
        local_inventory_Dict[(I,)] = []
        shipping_to_depot_Dict[(I,)] = []
        depot_repair_Dict[(I,)] = []
        shipping_to_base_Dict[(I,)] = []
    def __init__(self, location, part, ac):
        self.location = location
        self.part = part
        self.ac = ac
        self.ID = Spares.ID_count
        Spares.ID_count += 1
        if self.location == 'equip':
            Spares.equipped_Dict[(self.part,self.ac)].append(self)
            part_event_file.write(str(now()) + ',Part ' + str(self.
                ID) + ',created equipped' + '\n')
        elif self.location == 'spare':
            Spares.local_inventory_Dict[(self.part,)].append(self)
            part_event_file.write(str(now()) + ',Part ' + str(self.
                ID) + ',created spare' + '\n')
```

The following module is denoted by the number '8' in Figure 120. It defines behavior for both parts and aircraft simultaneously, and is used exclusively when aircraft have

requested parts from local inventory and not receieved them by the time the removal process would otherwise begin. In this case, the aircraft must wait until parts are available, and an external monitor is required to watch for available inventory and reactivate the aircraft once it has arrived.

```
class Inventory(Process):
    def __init__(self):
        Process.__init__(self)
    def Run(self):
        while 1:
            self.number_parts_needed = [0]*len(Input.MFHBR)
            for I in range(len(Fleet.PartQueue)):
                for J in range(len(Input.MFHBR)):
                    self.number_parts_needed[J] += Fleet.PartQueue[I
                       ].num_r[J] - Fleet.PartQueue[I].parts_filled[
                       J]
            for I in range(len(Input.MFHBR)):
                while len(Spares.local_inventory_Dict[(I,)]) > 0 and
                    self.number_parts_needed[I] > 0:
                    for J in range(len(Fleet.PartQueue)):
                        if Fleet.PartQueue[J].num_r[I] > 0:
                            if len(Spares.local_inventory_Dict[(I,)
                               ]) >= (Fleet.PartQueue[J].num_r[I] -
                               Fleet.PartQueue[J].parts_filled[I]):
                                for K in range(Fleet.PartQueue[J].
                                   num_r[I] - Fleet.PartQueue[J].
                                   parts_filled[I]):
                                    Spares.local_transit_Dict[(I,)].
                                       append(Spares.
                                       local_inventory_Dict[(I,)].
                                       pop())
                                    part_event_file.write(str(now())
                                        + ',Part ' + str(Spares.
                                       local_transit_Dict[(I,)][-1].
                                       ID) + ',sending part to
                                       maintenance facility' + '\n')
                                    for L in range(len(Input.MFHBR))
                                       :
                                        part_string = str(now()/24.)
                                            + ',Part category' + str
                                           (L+1)
                                        temp = 0
                                        for M in range(Input.num_ac)
                                           :
                                            temp += len(Spares.
                                               equipped_Dict[(L,M)])
                                        temp += len(Spares.
                                           local_transit_Dict[(L,)])
```

```
                    temp += len(Spares.
                        shipping_to_depot_Dict[(L
                        ,)])
                    temp += len(Spares.
                        shipping_to_base_Dict[(L
                        ,)])
                    temp2 = temp + len(Spares.
                        local_inventory_Dict[(L,)
                        ]) + len(Spares.
                        depot_repair_Dict[(L,)])
                    part_string = part_string +
                        ',' + str(len(Spares.
                        local_inventory_Dict[(L,)
                        ])) + ',' + str(len(
                        Spares.depot_repair_Dict
                        [(L,)])) + ',' + str(temp
                        ) + ',' + str(temp2) + '\
                        n'
                    part_file.write(part_string)
                self.number_parts_needed[I] -= 1
                Fleet.PartQueue[J].parts_filled[
                    I] += 1
            Fleet.PartQueue[J].parts_acquired[I]
                = 1
        elif len(Spares.local_inventory_Dict[(I
            ,)]) > 0:
            for K in range(len(Spares.
                local_inventory_Dict[(I,)])):
                Spares.local_transit_Dict[(I,)].
                    append(Spares.
                    local_inventory_Dict[(I,)].
                    pop())
                part_event_file.write(str(now())
                    + ',Part ' + str(Spares.
                    local_transit_Dict[(I,)][-1].
                    ID) + ',sending part to
                    maintenance facility' + '\n')
                for L in range(len(Input.MFHBR))
                    :
                    part_string = str(now()/24.)
                        + ',Part category' + str
                        (L+1)
                    temp = 0
                    for M in range(Input.num_ac)
                        :
                        temp += len(Spares.
                            equipped_Dict[(L,M)])
                    temp += len(Spares.
                        local_transit_Dict[(L,)])
                    temp += len(Spares.
                        shipping_to_depot_Dict[(L
                        ,)])
```

```
                                    temp += len(Spares.
                                        shipping_to_base_Dict[(L
                                        ,)])
                                    temp2 = temp + len(Spares.
                                        local_inventory_Dict[(L,)
                                        ]) + len(Spares.
                                        depot_repair_Dict[(L,)])
                                    part_string = part_string +
                                        ',' + str(len(Spares.
                                        local_inventory_Dict[(L,)
                                        ])) + ',' + str(len(
                                        Spares.depot_repair_Dict
                                        [(L,)])) + ',' + str(temp
                                        ) + ',' + str(temp2) + '\
                                        n'
                                    part_file.write(part_string)
                                self.number_parts_needed[I] -= 1
                                Fleet.PartQueue[J].parts_filled[
                                    I] += 1
            if len(Fleet.PartQueue) > 0:
                I_delete = 0
                for I in range(len(Fleet.PartQueue)):
                    if sum(Fleet.PartQueue[I-I_delete].
                        parts_acquired) == len(Input.MFHBR):
                         reactivate(Fleet.PartQueue[I-I_delete])
                         Fleet.PartQueue.pop(I-I_delete)
                         I_delete += 1
            yield hold, self, 0.25
```

The following module is denoted by the number '9' in Figure 120. Where the module Spares was used to hold the part objects in different categories, the Depot module is used to define the behavior that moves part objects between different states. It is activated once parts are removed from aircraft and manages the logic associated with sending them to the depot for repair, performing the repair, and returning them to inventory. This is implemented as a separate module due to the delays involved, since the logic must pause within the simulation in between different part state transitions. If this were implemented as part of the fleet logic, the aircraft would need to wait while parts are sent away and back before continuing on with repairs.

```
class Depot(Process):
    ID_count = 1
    def __init__(self, part, ac):
        Process.__init__(self)
```

```
        self.part = part
        self.ac = ac
        self.ID = Depot.ID_count
        Depot.ID_count += 1
    def Run(self):
        #Send one part through the depot repair cycle
        ##Change state to shipping to depot
        Spares.shipping_to_depot_Dict[(self.part,)].append(Spares.
            equipped_Dict[(self.part,self.ac)].pop())
        part_event_file.write(str(now()) + ',Part ' + str(Spares.
            shipping_to_depot_Dict[(self.part,)][-1].ID) + ',shipping
             to depot' + '\n')
        for I in range(len(Input.MFHBR)):
            part_string = str(now()/24.) + ',Part category' + str(I
                +1)
            temp = 0
            for J in range(Input.num_ac):
                temp += len(Spares.equipped_Dict[(I,J)])
            temp += len(Spares.local_transit_Dict[(I,)])
            temp += len(Spares.shipping_to_depot_Dict[(I,)])
            temp += len(Spares.shipping_to_base_Dict[(I,)])
            temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) +
                len(Spares.depot_repair_Dict[(I,)])
            part_string = part_string + ',' + str(len(Spares.
                local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
                depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' +
                str(temp2) + '\n'
            part_file.write(part_string)
        ##Hold for delay while shipping to depot
        yield hold, self, random.uniform(Assumption.
            step_ship_base_dep_min, Assumption.step_ship_base_dep_max
            )
        ##Change state to being repaired at depot
        Spares.depot_repair_Dict[(self.part,)].append(Spares.
            shipping_to_depot_Dict[(self.part,)].pop())
        part_event_file.write(str(now()) + ',Part ' + str(Spares.
            depot_repair_Dict[(self.part,)][-1].ID) + ',depot repair'
             + '\n')
        for I in range(len(Input.MFHBR)):
            part_string = str(now()/24.) + ',Part category' + str(I
                +1)
            temp = 0
            for J in range(Input.num_ac):
                temp += len(Spares.equipped_Dict[(I,J)])
            temp += len(Spares.local_transit_Dict[(I,)])
            temp += len(Spares.shipping_to_depot_Dict[(I,)])
            temp += len(Spares.shipping_to_base_Dict[(I,)])
            temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) +
                len(Spares.depot_repair_Dict[(I,)])
            part_string = part_string + ',' + str(len(Spares.
                local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
                depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' +
                str(temp2) + '\n'
            part_file.write(part_string)
```

```
##Hold for delay while repairing at depot
yield hold, self, random.triangular(Assumption.
   step_D_rep_min, Assumption.step_D_rep_max, Assumption.
   step_D_rep_mod)
##Change state to shipping to base
Spares.shipping_to_base_Dict[(self.part,)].append(Spares.
   depot_repair_Dict[(self.part,)].pop())
part_event_file.write(str(now()) + ',Part ' + str(Spares.
   shipping_to_base_Dict[(self.part,)][-1].ID) + ',shipping
   to base' + '\n')
for I in range(len(Input.MFHBR)):
    part_string = str(now()/24.) + ',Part category' + str(I
       +1)
    temp = 0
    for J in range(Input.num_ac):
        temp += len(Spares.equipped_Dict[(I,J)])
    temp += len(Spares.local_transit_Dict[(I,)])
    temp += len(Spares.shipping_to_depot_Dict[(I,)])
    temp += len(Spares.shipping_to_base_Dict[(I,)])
    temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) +
       len(Spares.depot_repair_Dict[(I,)])
    part_string = part_string + ',' + str(len(Spares.
       local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
       depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' +
       str(temp2) + '\n'
    part_file.write(part_string)
##Hold for delay while shipping to base
yield hold, self, random.triangular(Assumption.
   step_ship_dep_base_min, Assumption.step_ship_dep_base_max
   , Assumption.step_ship_dep_base_mod)
##Change state to local inventory
Spares.local_inventory_Dict[(self.part,)].append(Spares.
   shipping_to_base_Dict[(self.part,)].pop())
part_event_file.write(str(now()) + ',Part ' + str(Spares.
   local_inventory_Dict[(self.part,)][-1].ID) + ',local
   inventory' + '\n')
for I in range(len(Input.MFHBR)):
    part_string = str(now()/24.) + ',Part category' + str(I
       +1)
    temp = 0
    for J in range(Input.num_ac):
        temp += len(Spares.equipped_Dict[(I,J)])
    temp += len(Spares.local_transit_Dict[(I,)])
    temp += len(Spares.shipping_to_depot_Dict[(I,)])
    temp += len(Spares.shipping_to_base_Dict[(I,)])
    temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) +
       len(Spares.depot_repair_Dict[(I,)])
    part_string = part_string + ',' + str(len(Spares.
       local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
       depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' +
       str(temp2) + '\n'
    part_file.write(part_string)
```

The following lines of code are used to record data for output once per simulation day. This is where additional output metrics could be defined and computed from other parameters within the code.

```
class Daily(Process):
    average_PT = 6.8
    average_MT = 9.8
    def __init__(self):
        Process.__init__(self)
    def Run(self):
        while 1:
            if now()/24. == 0:
                ao_file.write('Day, Ao, \n')
                ro_file.write('Day, Ro, \n')
                mis_prep_time_len = 0
                mis_all_time_len = 0
            if now()%24. == 0 and now()/24. > 0:
                old_mis_prep_time_len = 0
                old_mis_prep_time_len += mis_prep_time_len
                mis_prep_time_len = len(Fleet.mis_prep_avg)
                old_mis_all_time_len = 0
                old_mis_all_time_len += mis_all_time_len
                mis_all_time_len = len(Fleet.mis_all_avg)
                try:
                    Daily.average_PT = sum(Fleet.mis_prep_avg[
                       old_mis_prep_time_len:mis_prep_time_len])/(
                       mis_prep_time_len-old_mis_prep_time_len)
                    Daily.average_MT = sum(Fleet.mis_all_avg[
                       old_mis_all_time_len:mis_all_time_len])/(
                       old_mis_all_time_len-mis_all_time_len)
                except:
                    dummy = 0
                for I in range(Input.num_ac):
                    if Fleet.AC[I].state == 0:
                        Fleet.AC[I].available_time += now() - Fleet.
                           AC[I].t_prev
                    elif Fleet.AC[I].state == 1:
                        Fleet.AC[I].flying_time += now() - Fleet.AC[
                           I].t_prev
                    elif Fleet.AC[I].state == 2:
                        Fleet.AC[I].bar_time += now() - Fleet.AC[I].
                           t_prev
                    elif Fleet.AC[I].state == 3:
                        Fleet.AC[I].bap_time += now() - Fleet.AC[I].
                           t_prev
                    elif Fleet.AC[I].state == 4:
                        Fleet.AC[I].bbr_time += now() - Fleet.AC[I].
                           t_prev
                    Fleet.AC[I].t_prev = now()
                #Record the day's Ao
```

```
        available_sum = 0
        flying_sum = 0
        bar_sum = 0
        bap_sum = 0
        bbr_sum = 0
        available_string = str(now())
        flying_string = str(now())
        bar_string = str(now())
        bap_string = str(now())
        bbr_string = str(now())
        temp_avail = 0
        temp_fly = 0
        temp_bar = 0
        temp_bap = 0
        temp_bbr = 0
        for I in range(Input.num_ac):
            available_sum += Fleet.AC[I].available_time
            flying_sum += Fleet.AC[I].flying_time
            bar_sum += Fleet.AC[I].bar_time
            bap_sum += Fleet.AC[I].bap_time
            bbr_sum += Fleet.AC[I].bbr_time
            temp_avail += Fleet.AC[I].available_time
            temp_fly += Fleet.AC[I].flying_time
            temp_bar += Fleet.AC[I].bar_time
            temp_bap += Fleet.AC[I].bap_time
            temp_bbr += Fleet.AC[I].bbr_time
            Fleet.AC[I].available_time = 0
            Fleet.AC[I].flying_time = 0
            Fleet.AC[I].bar_time = 0
            Fleet.AC[I].bap_time = 0
            Fleet.AC[I].bbr_time = 0
        ao = (available_sum + flying_sum)/(available_sum +
           flying_sum + bar_sum + bap_sum + bbr_sum)
        string = str(now()/24.) + ',' + str(ao) + ', \n'
        all_string = str(now()/24.) + ',' + str(temp_avail)
           + ',' + str(temp_fly) + ',' + str(temp_bar) + ','
            + str(temp_bap) + ',' + str(temp_bbr) + '\n'
        ao_file.write(string)
        for J in range(len(Input.surge_level)):
            if now() > Input.surge_time[J] and now() < Input
               .surge_time[J+1]:
                ro = 1-(SortieGen.sortie_backlog[-1]/(Input.
                   OT[int(((now()-24.)/24.)%7)]*Input.
                   surge_level[J]))
        string2 = str(now()/24.) + ',' + str(ro) + ', \n'
        ro_file.write(string2)
    print now()/24., SortieGen.sortie_backlog, SortieGen.
       sortie_aborted
    yield hold, self, 24.
```

The following lines of code are the main function. It initializes and calls all the other

modules within the code in order, and then sets the simulation to run. After the simulation has been completed, several final outputs are printed for review.

```
def main():
    initialize()
    if ModelVersion.PHM == 1:
        for I in range(Input.num_ac):
            MS = MaintenanceSchedule()
            activate(MS,MS.Run())
    SB = SortieBegin()
    activate(SB,SB.Run())
    SG = SortieGen()
    activate(SG,SG.Run())
    FC = FlightChief()
    GC = GroundCrew()
    M = Maintenance()
    R = Runway()
    for I in range(Input.num_ac):
        F = Fleet()
        activate(F,F.Run())
    S_Dict = {}
    for I in range(len(Input.MFHBR)):
        for J in range(Input.num_ac):
            for K in range(1):
                S_Dict[str(J*K)] = Spares('equip', I, J)
        part_string = str(now()/24.) + ',Part category' + str(I+1)
        temp = 0
        for J in range(Input.num_ac):
            temp += len(Spares.equipped_Dict[(I,J)])
        temp += len(Spares.local_transit_Dict[(I,)])
        temp += len(Spares.shipping_to_depot_Dict[(I,)])
        temp += len(Spares.shipping_to_base_Dict[(I,)])
        temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) + len(
           Spares.depot_repair_Dict[(I,)])
        part_string = part_string + ',' + str(len(Spares.
           local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
           depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' + str(
           temp2) + '\n'
        part_file.write(part_string)
        for J in range(Input.num_spares[I]):
            S_Dict[str(J+Input.num_ac)] = Spares('spare', I, 0)
        part_string = str(now()/24.) + ',Part category' + str(I+1)
        temp = 0
        for J in range(Input.num_ac):
            temp += len(Spares.equipped_Dict[(I,J)])
        temp += len(Spares.local_transit_Dict[(I,)])
        temp += len(Spares.shipping_to_depot_Dict[(I,)])
        temp += len(Spares.shipping_to_base_Dict[(I,)])
        temp2 = temp + len(Spares.local_inventory_Dict[(I,)]) + len(
           Spares.depot_repair_Dict[(I,)])
```

```
            part_string = part_string + ',' + str(len(Spares.
               local_inventory_Dict[(I,)])) + ',' + str(len(Spares.
               depot_repair_Dict[(I,)])) + ',' + str(temp) + ',' + str(
               temp2) + '\n'
            part_file.write(part_string)
        I = Inventory()
        activate(I,I.Run())
        Da = Daily()
        activate(Da,Da.Run())
        simulate(until=Input.sim_time)
        mis_prep_avg = sum(Fleet.mis_prep_avg)/len(Fleet.mis_prep_avg)
        mis_prep_file.write(str(Fleet.mis_prep_min) + ',' + str(Fleet.
           mis_prep_max) + ',' + str(mis_prep_avg) + ',' + str(Fleet.
           mis_prep_max-Fleet.mis_prep_min))
        print Output.num_broken, Output.num_used, Output.
           part_life_wasted
        print Output.negative_prediction_times, Output.
           missed_prediction_times
        print Fleet.maintenance_times
        print Fleet.maintenance_aircraft
        print SortieGen.total_missions_generated
        print Fleet.total_missions_flown
        print len(Fleet.maintenance_times), sum(Output.num_used)
        print Fleet.maintenance_numbers
        print Fleet.maintenance_numbers_after
        print Fleet.num_detections
        print Fleet.mission_failures
        print Fleet.phm_repairs
        print Fleet.flight_hours
        print Fleet.low_detect
        print Fleet.one_count,Fleet.two_count

if __name__ == '__main__': main()
```

REFERENCES

[1] "Interim dod instruction 5000.02: Operation of the defense acquisition system."

[2] Abell, J. B., Carter, G. M., Isaacson, K. E., and Lippiatt, T. F., "Estimating requirements for aircraft recoverable spare and depot repair," tech. rep., RAND, 1993.

[3] Adams, J. L., Abell, J. B., and Isaacson, K. E., "Modeling and forecasting the demand for aircraft recoverable spare parts," tech. rep., RAND, 1993.

[4] Akturk, M. S. and Gurel, S., "Machining conditions-based preventive maintenance," *International Journal of Production Research*, vol. 45, pp. 1725–1743, April 2007.

[5] Allen, T. M., "Zero maintenance - definition." Presentation.

[6] Ames, W. J., "Logistical effectiveness of two-level maintenance," tech. rep., Air Command and Staff College Air University, 2000.

[7] Aronis, K.-P., Magou, I., Dekker, R., and Tagaras, G., "Inventory control of spare parts using a bayesian approach: A case study," *European Journal of Operational Research*, vol. 154, pp. 730–739, 2004.

[8] Ayyub, B. M. and Klir, G. J., *Uncertainty Modeling and Analysis in Engineering and the Sciences*. CRC Press, 2006.

[9] Bae, S. J., Kuo, W., and Kvam, P. H., "Degradation models and implied lifetime distributions," *Reliability Engineering and System Safety*, vol. 92, pp. 601–608, 2007.

[10] Barlow, R. and Hunter, L., "Optimum preventive maintenance policies," *Operations Research*, vol. 8, pp. 90–100, Jan/Feb 1960.

[11] Bateman, J. F., "Preventive maintenance: Stand alone manufacturing compared with cellular manufacturing," *Industrial Management*, vol. 20, pp. 19–21, January/February 1995.

[12] Birge, J. R. and Louveaux, F., *Introduction to Stochastic Programming*. Springer-Verlag, 1997.

[13] Boggs, P. T. and Tolle, J. W., "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[14] Boito, M., Cook, C. R., and Graser, J. C., "Contractor logistics support in the u.s. air force," tech. rep., RAND, 2009.

[15] Booz Allen, "Performance based logistics." Electronic.

[16] Brown, F. T., *Engineering System Dynamics: A Unified Graph-Centered Approach*. Taylor & Francis Group, 2nd ed., 2007.

[17] Caramia, M. and Dell'Olmo, P., *Multi-objective Management in Freight Logistics Increasing Capacity, Service Level and Safety with Optimization Algorithms*, ch. 2, pp. 11–36. Springer, 2008.

[18] Chin, H. H., "Turbine engine hot section prognostics," *Applied Concept Research Inc.*, 2003.

[19] Chinnam, R. B., "On-line reliability estimation for individual components using statistical degradation signal models," *Quality and Reliability Engineering International*, vol. 18, pp. 53–73, 2002.

[20] Cho, P. Y., "Optimal scheduling of fighter aircraft maintenance," Master's thesis, Massachusetts Institute of Technology, 2009.

[21] Chueng, K. L. and Hausman, W. H., "Joint determination of preventive maintenance and safety stocks in an unreliable production environment," *Naval Research Logistics*, vol. 44, pp. 257–272, 1997.

[22] Cole, A., "Pentagon's carter eyes closer industry ties," *Wall Street Journal, Eastern edition*, May 1 2009.

[23] Crocker, J. and Kumar, U., "Age-related maintenance versus reliability centred maintenance: a case study on aero-engines," *Reliability Engineering and System Safety*, vol. 67, pp. 113–118, 2000.

[24] Danis, M., "Learning algorithms in optimization of project scheduling in microsoft project 2003," Master's thesis, Stockholm University, 2005.

[25] Daskilewicz, M. J. and German, B. J., "Nondomination-level coordinate system for parameterizing and exploring pareto frontiers," *AIAA Journal*, vol. 51, no. 8, pp. 1946–1959, 2013.

[26] Defense Acquisition University, *Performance Based Logistics: A Program Manager's Product Support Guide*. Department of Defense, 2005.

[27] Defense Acquisition University, "Information paper on materiel availability (am), materiel reliability (rm), and operational availability (ao)." Online, December 2008.

[28] Dekker, R., "Applications of maintenance optimization models: a review and analysis," *Reliability Engineering and System Safety*, vol. 51, pp. 229–240, 1996.

[29] Dekker, R. and Scarf, P. A., "On the impact of optimisation models in maintenance decision making: the state of the art," *Reliability Engineering and System Safety*, vol. 60, pp. 111–119, 1998.

[30] Department of the Air Force, "Contract sustainment support guide," tech. rep., United States Air Force, 2013.

[31] Department of the Army, *Department of the Army Technical Bulletin: Phase Maintenance System for Army Aircraft*. Department of the Army, May 1984.

[32] Doksum, K. A. and Hyland, A., "Models for variable-stress accelerated life testing experiments based on wiener processes and the inverse gaussian distribution," *Technometrics*, vol. 34, pp. 74–82, 1992.

[33] Ehrgott, M., *Multicriteria Optimization*. Springer Berlin Heidelberg New York, 2005.

[34] Ehrgott, M. and Gandibleux, X., "A survey and annotated bibliography of multiobjective combinatorial optimization," *OR Spektrum*, vol. 22, pp. 425–460, 2000.

[35] Ehrgott, M. and Gandibleux, X., "Approximate solution methods for multiobjective combinatorial optimization," *Top*, vol. 12, no. 1, pp. 1–89, 2004.

[36] Eisenmann Sr., R. C. and C., E. J. R., *Machinery malfunction diagnosis and correction: vibration analysis and troubleshooting for the process industries*. Prentice-Hall, 1998.

[37] Elwany, A. H. and Gebraeel, N. Z., "Sensor-driven prognostic models for equipment replacement and spare parts inventory," *IIE Transactions*, vol. 40, pp. 629–639, April 2008.

[38] Elwany, A. H., Gebraeel, N. Z., and Maillart, L. M., "Structured replacement policies for components with complex degradation processes and dedicated sensors," *Operations Research*, vol. 59, no. 3, pp. 684–695, 2011.

[39] Faas, P. D., "Simulation of autonomic logistics system (als) sortie generation," Master's thesis, Air Force Institute of Technology, 2003.

[40] Ferguson, T. S., "Linear programming: A concise introduction." Course notes found online.

[41] Fishman, G. S., *Discrete-Event Simulation: Modeling, Programming and Analysis*. Springer, 2001.

[42] Fraser, K. F., "An overview of health and usage monitoring systems (hums) for military helicopters," tech. rep., Defence Science and Technology Organziation Melbourne (Australia), 1994.

[43] Gaguzis, M. P., "Effectiveness of condition-based maintenance in army aviation," Master's thesis, U. S. Army Command and General Staff College, 2009.

[44] Gebraeel, N. Z. and Lawley, M. A., "A neural network degradation model for computing and updating residual life distributions," *IEEE Transactions on Automation Science and Engineering*, vol. 5, pp. 154–163, January 2008.

[45] Gebraeel, N. Z., Lawley, M. A., and Li, Rong; Ryan, J. K., "Residual-life distributions from component degradation signals a bayesian approach," *IIE Transactions*, vol. 37, pp. 543–557, 2005.

[46] German, R., Logothetis, D., and Trivedi, K. S., "Transient analysis of markov regenerative stochastic petri nets: A comparison of approaches," in *Proceedings of the Sixth International Workshop on Petri nets and Performance Models*, 1995.

[47] Gilbert, N. and Terna, P., "How to build and use agent-based models in social science," *Mind & Society*, vol. 1, pp. 57–72, 2000.

[48] Golmakani, H. R., "Condition-based inspection scheme for condition-based maintenance," *International Journal of Production Research*, vol. 50, pp. 3920–3935, 2012.

[49] Grall, A., Brenguer, C., and Dieulle, L., "A condition-based maintenance policy for stochastically deteriorating systems," *Reliability Engineering and System Safety*, vol. 76, pp. 167–180, 2002.

[50] Gurobi, *Optimizing MIP Problems: A primer on the basics of mixed integer programming.*

[51] Haas, P. J., *Stochastic Petri Nets: Modeling, Stability, Simulation.* Springer-Verlag, 2002.

[52] Hartzell, A. L., Da Silva, M. G., and Shea, H. R., *MEMS Reliability.* Springer, 2010.

[53] Hess, A., Calvello, G., and Dabney, T., "Phm a key enabler for the jsf autonomic logistics support concept," in *2004 IEEE Aerospace Conference Proceedings*, pp. 3543–3550, 2004.

[54] Hill, R. R. and McIntyre, G. A., "Applications of discrete event simulation modeling to military problems," in *Proceedings of the 2001 Winter Simulation Conference* (Peters, B. A., Smith, J. S., Medeiros, D. J., and Rohrer, M. W., eds.), 2001.

[55] Hillier, F. S. and Lieberman, G. J., *Introduction to Operations Research.* Holden-Day, 7th ed., 1967.

[56] Iakovidis, K., "Comparing f-16 maintenance scheduling philosophies," Master's thesis, Air Force Institute of Technology, 2005.

[57] Jardine, A. K. S., Banjevic, D., and Makis, V., "Optimal replacement policy and the structure of software for condition-based maintenance," *Journal of Quality in Maintenance Engineering*, vol. 3, no. 2, pp. 109–119, 1997.

[58] Jardine, A. K. S., Joseph, T., and Banjevic, D., "Optimizing condition-based maintenance decision for equipment subject to vibration monitoring," *Journal of Quality in Maintenance Engineering*, vol. 5, no. 3, pp. 192–202, 1999.

[59] Jardine, A. K. S., *Handbook of Operations Research*, ch. I-12, pp. 372–389. Van Nostrand Reinhold Company, 1978.

[60] Jardine, A. K. S., Lin, D., and Banjevic, D., "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, pp. 1483–1510, 2006.

[61] Jardine, A. K. S., *Maintenance Excellence: Optimizing Equipment Life-Cycle Decisions*, ch. 11, pp. 289–322. Marcel Dekker, 2001.

[62] Kang, K., Gue, K. R., and Eaton, D. R., "Cycle time reduction for naval aviation depots," in *Proceedings of the 1998 Winter Simulation Conference* (Medeiros, D., Watson, E., Carson, J., and Manivannan, M., eds.), 1998.

[63] Kharoufeh, J. P. and Cox, S. M., "Stochastic models for degradation-based reliability," *IIE Transactions*, vol. 37, pp. 533–542, 2005.

[64] Kharoufeh, J. P., Solo, C. J., and Ulukus, M. Y., "Semi-markov models for degradation-based reliability," *IIE Transactions*, vol. 42, pp. 599–612, 2010.

[65] Kiebler, K. K., Dibble, G. B., Klapper, L. S., Linville, R. P., Perry, J. H., and Zurlo, J. M., "The depot repair cycle process: Opportunities for business practice improvement," tech. rep., Logistics Management Institute, 2000.

[66] Klutke, G.-A., Kiessler, P. C., and Wortman, M. A., "A critical look at the bathtub curve," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 125–129, 2003.

[67] Koochaki, J., Bokhorst, J. A., Hans, W., and Warse, K., "The influence of condition-based maintenance on workforce planning and maintenance scheduling," *International Journal of Production Research*, vol. 51, pp. 2339–2351, 2013.

[68] Koochaki, J., Bokhorst, J. A., Wortmann, H., and Klingenberg, W., "Condition based maintenance in the context of opportunistic maintenance," *International Journal of Production Research*, vol. 50, pp. 6918–6929, 2012.

[69] Krieg, K. J., "Dodi 3110.05," September 2006.

[70] Kumar, U. D., "Tutorials on life cycle coscost and reliability engineering: Course material." Course notes provided by Dr. Schrage.

[71] Kumar, U. D., "New trends in aircraft reliability and maintenance measures," *Journal of Quality in Maintenance Engineering*, vol. 5, no. 4, pp. 287–296, 1999.

[72] Lawler, G. F., *Introduction to Stochastic Processes*. Chapman & Hall/CRC, second ed., 2006.

[73] Lee, H. L., Padmanbhan, V., and Whang, S., "Information distortion in a supply chain: The bullwhip effect," *Management Science*, vol. 43, no. 4, pp. 546–558, 1997.

[74] Lesobre, R., Bouvard, K., Brenguer, C., Barros, A., and Cocquempot, V., "A maintenance free operating period policy for a multi-component system with different information levels on the components state," *Chemical Engineering Transactions*, vol. 33, pp. 1051–1056, 2013.

[75] Liu, P. H., Makis, V., and Jardine, A. K. S., "Scheduling of the optimal tool replacement time in a flexible manufacturing system," *IIE Transactions*, vol. 33, pp. 487–495, 2001.

[76] Lockheed Martin Aeronautics Company, "Usaf extends phase maintenance interval on lockheed martin block 40/42/50/52 f-16s usaf extends phase maintenance interval on lockheed martin block 40/42/50/52 f-16s." Press Release, May 24 2000.

[77] Luenberger, D. G. and Ye, Y., *Linear and Nonlinear Programming*. Springer Science+Business Media LLC, 3rd ed., 2008.

[78] Malley, M. E., "A methodology for simulating the joint strike fighter's (jsf) prognostics and heath management system," Master's thesis, Air Force Institute of Technology, 2001.

[79] Martin, L., "F-35 lightning ii: Defining the future."

[80] Mathaisel, D. F., Cathcart, T. P., and Comm, C. L., "A framework for benchmarking, classifying, and implementing best sustainment practices," *Benchmarking: An International Journal*, vol. 11, no. 4, pp. 403–417, 2004.

[81] Matloff, N., *Introduction to Discrete-Event Simulation and the SimPy Language*. University of California, Davis, 2008.

[82] Mavrotas, G. and Diakoulaki, D., "A branch and bound algorithm for mixed zero-one multiple objective linear programming," *European Journal of Operational Research*, vol. 107, pp. 530–541, 1998.

[83] Meeker, W. Q., Escobar, L. A., and Lu, C. J., "Accelerated degradation tests: Modeling and analysis," *Technometrics*, vol. 40, pp. 89–99, May 1998.

[84] Mobley, R. K., *An Introduction to Predictive Maintenance*. Elsevier, 2002.

[85] Muckstadt, J. A., *Analysis and Algorithms for Service Parts Supply Chains*. Springer, 2005.

[86] Mullen, M., "Joint publication 3-0: Joint operations," 2011.

[87] Neelakanteswararao, A. and Bhadury, B., "Opportunistic maintenance of multi-equipment system: A case study," *Quality and Reliability Engineering International*, vol. 16, pp. 487–500, 2000.

[88] Nelson, W., *Accelerated testing: Statistical models, test plans and data analyses*. Wiley-Interscience, 2004.

[89] Ng, I. C., Maull, R., and Yip, N., "Outcome-based contracts as a driver for systems thinking and service-dominant logic in service science: Evidence from the defence industry," *European Management Journal*, vol. 27, pp. 377–387, 2009.

[90] Niazi, M. and Hussain, A., "Agent-based computing from multi-agent systems to agent-based models, a visual survey," *Scientometrics*, vol. 89, pp. 479–499, 2011.

[91] Office of the Undersecretary of Defense (Comptroller), "Operation and maintenance programs (o-1) revolving and management funds (rf-1)," tech. rep., United States Department of Defense, 2012.

[92] Olfati-Saber, R., "Concensus and cooperation in nenetwork multi-agent systems," in *Proceedings of the IEEE*, vol. 95, pp. 215–233, 2007.

[93] Randall, W. S., Nowicki, D. R., and Hawkins, T. G., "Explaining the effectiveness of performance-based logistics: a quantitative examination," *The International Journal of Logistics Management*, vol. 22, no. 3, pp. 324–348, 2011.

[94] Relf, M. N., "Maintenance-free operating periods - the designer's challenge," *Quality and Reliability Engineering International*, vol. 15, pp. 111–116, 1999.

[95] Robinson, S., "Discrete-event simulation: from the pioneers to the present, what next?," *Journal of the Operational Research Society*, vol. 56, pp. 619–629, 2005.

[96] Rushmeier, R. A. and Kontogiorgis, S. A., "Advances in the optimization of airline fleet assignment," *Transportation Science*, vol. 31, no. 2, pp. 159–169, 1997.

[97] Sadoun, B., "Applied system simulation: a review study," *Information Sciences*, vol. 124, pp. 173–192, 2000.

[98] Saltmarsh, E. A. and Mavris, D. N., "Simulating corrective maintenance: Aggregating component level maintenance time uncertainty at the system level," in *2013 Conference on Systems Engineering Research* (Paredis, C. J., Bishop, C., and Bodner, D., eds.), vol. 16, pp. 459–468, Elsevier, 2013.

[99] Scarf, P. A., "On the application of mathematical models in maintenance," *European Journal of Operational Research*, vol. 99, pp. 493–506, 1997.

[100] Scheuren, W., "Safety and the military aircraft joint strike fighter prognostics and health management," in *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1998.

[101] Schrijver, A., *Theory of Linear and Integer Programming*. John Wiley & Sons, Ltd., 1986.

[102] Schweitzer, P. J., "Optimal replacement policies for hyperexponentially and uniformly distributed lifetimes," *Operations Research*, vol. 15, pp. 360–362, March 1967.

[103] Seibers, P.-O., "Simulation: A key technique in operational research." Online, February 2006. Research seminar presentation.

[104] Shah, J., *Supply Chain Management: Text and Cases*. Pearson Education, 2009.

[105] Swanson, D. C., "A general prognostic tracking algorithm for predictive maintenance," in *Proceedings in IEEE Aerospace Conference*, vol. 6, pp. 62971–62977, 2001.

[106] Swanson, L., "Linking maintenance strategies to performance," *International Journal of Production Economics*, vol. 70, pp. 237–244, 2001.

[107] Tako, A. A. and Robinson, S., "The application of discrete event simulation and system dynamics in the logistics and supply chain context," *Decision Support Systems*, vol. 52, pp. 802–815, 2012.

[108] Tompkins, M. F., "Optimization techniques for task allocation and scheduling in distributed multi-agent operations," Master's thesis, Massachusetts Institute of Technology, 2003.

[109] Tripp, R. S., McGarvey, R. G., Van Roo, B. D., Masters, J. M., and Sollinger, J. M., "A repair network concept for air force maintenance: Conclusions from analysis of c-130, f-16, and kc-135 fleets," tech. rep., RAND, 2010.

[110] United States Air Force, *Aircraft and Equipment Maintenance Management*. United States Air Force, October 2013.

[111] United States Air Force Scientific Advisory Board, "Sustaining air force aging aircraft into the 21st century," tech. rep., United States Air Force, 2011.

[112] United States Department of Defense, "Dod weapon system acquisition reform product support assessment," tech. rep., November 2009.

[113] United States Department of Defense, "Contract.." Press Release, December 2012.

[114] United States Government Accountability Office, "Inventory management: The army could reduce logistics costs for aviation parts by adopting best practices," tech. rep., United States Government Accountability Office, 1997.

[115] United States Government Accountability Office, "Defense inventory: Defense logistics agency needs to expand on efforts to more effectively manage spare parts," Tech. Rep. GAO-10-496, United States Government Accountability Office, May 2010.

[116] United States Government Accountability Office, "Joint strike fighter: Dod actions needed to further enhance restructuring and address affordability risks," tech. rep., United States Government Accountability Office, 2012.

[117] United States Under Secretary of Defense, "Performance based logistics: Purchasing under performance based criteria," 2004.

[118] Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design: With Applications*. McGraw-Hill Companies, 1984.

[119] Wiseman, M., *Maintenance Excellence: Optimizing Equipment Life-Cycle Decisions*, ch. 12, pp. 323–366. Marcel Dekker, 2001.

[120] Wood, B., "Intelligent building care," *Facilities*, vol. 17, pp. 189–194, 1999.

[121] Wu, S.-j., Gebraeel, N., Lawley, M. A., and Yih, Y., "A neural network integrated decision support system for condition-based optimal predictive maintenance policy," *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, vol. 37, pp. 226–236, March 2007.

[122] Xia, T., Xi, L., Zhou, X., and Lee, J., "Condition-based maintenance for intelligent monitored series system with independent machine failure modes," *International Journal of Production Research*, vol. 51, pp. 4585–4596, 2013.

[123] Yager, N. A., "Models for sortie generation with autonomic logistics capabilities," Master's thesis, Air Force Institute of Technology, 2003.

[124] Zhou, M. and Venkatesh, K., *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. World Scientific Publishing Co., 1999.

VITA

Elizabeth Saltmarsh grew up in North Pole, Alaska where she attended North Pole High School, graduating in 2004. She completed her Bachelor of Science in Aerospace Engineering at Georgia Institute of Technology in 2008. After graduating, she began working at the Aerospace Systems Design Laboratory as a research assistant during which time she completed her Master of Science in Aerospace Engineering in 2010. After completing her Doctor of Philosophy, she will begin working for Northrop Grumman as part of their Future Technical Leaders program.