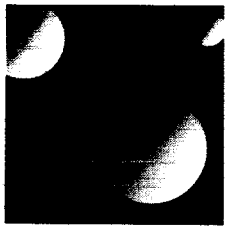
NAG1-01025

RELIABILITY AND PRODUCTIVITY MODELING FOR THE OPTIMIZATION OF SEPARATED SPACECRAFT INTERFEROMETERS

Julie Wertz, David Miller

May, 2002

SSL Report # 9-02



MIT
Space
Engineering
Research
Center

Massachusetts
Institute of
Technology

Cambridge
Massachusetts
02139

Reliability and Productivity Modeling for the Optimization of Separated Spacecraft Interferometers

by

Julie Wertz

S.B., Aeronautical and Astronautical Engineering
Massachusetts Institute of Technology, 2000

SUBMITTED TO THE DEPARTMENT OF AERONAUTICAL AND ASTRONAUTICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Signature of Author .....
Department of Aeronautics and Astronautics
May 24, 2002

Certified by .....
David W. Miller
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Wallace E. Vander Velde
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Reliability and Productivity Modeling for the Optimization of Separated Spacecraft Interferometers

by

JULIE WERTZ

Submitted to the Department of Aeronautics and Astronautics
on May 24, 2002 in Partial Fulfillment of the
Requirements for the Degree of Master of Science
at the Massachusetts Institute of Technology

ABSTRACT

As technological systems grow in capability, they also grow in complexity. Due to this complexity, it is no longer possible for a designer to use engineering judgement to identify the components that have the largest impact on system life cycle metrics, such as reliability, productivity, cost, and cost effectiveness. One way of identifying these key components is to build quantitative models and analysis tools that can be used to aid the designer in making high level architecture decisions. Once these key components have been identified, two main approaches to improving a system using these components exist: add redundancy or improve the reliability of the component. In reality, the most effective approach to almost any system will be some combination of these two approaches, in varying orders of magnitude for each component. Therefore, this research tries to answer the question of how to divide funds, between adding redundancy and improving the reliability of components, to most cost effectively improve the life cycle metrics of a system. While this question is relevant to any complex system, this research focuses on one type of system in particular: Separate Spacecraft Interferometers (SSI). Quantitative models are developed to analyze the key life cycle metrics of different SSI system architectures. Next, tools are developed to compare a given set of architectures in terms of total performance, by coupling different life cycle metrics together into one performance metric. Optimization tools, such as simulated annealing and genetic algorithms, are then used to search the entire design space to find the "optimal" architecture design. Sensitivity analysis tools have been developed to determine how sensitive the results of these analyses are to uncertain user defined parameters. Finally, several possibilities for the future work that could be done in this area of research are presented.

Thesis Supervisor:
Prof. David W. Miller
Dept. of Aeronautics and Astronautics

ACKNOWLEDGMENTS

This research was funded by both the Langley Research Center's Risk Based Design Program, techinical monitor Sean Kenny, and the Lockheed Martin Terrestrial Planet Finder Team, technical monitor Domenick Tenerelli.

There are several people I would like to thank for their direct impact on this work:

- David Miller, my advisor, for all his work and support. His love of engineering and problem solving is always inspiring.
- Cyrus Jilla, for being a great mentor. Thanks for all the help and support you've given me throughout the past two years.
- The JPL StarLight team, specifically Oliver Lay, Brian Barden, and Gary Blackwood, for sharing their technical knowledge with me and giving me a great learning experience.
- Edmund Kong, Alice Liu, Becky Masterson, and Allen Chen for their technical advice.

In addition, there are several people I would like to thank for all of the support they gave me during this work:

- The SSL community, for all of their friendships. I feel very lucky to have so much fun with the people I work with.
- My quals studying buddies, Shannon and Al, for making me laugh, "getting this party started", and keeping me going throughout January. You're two of the most incredible people I have ever met, both intellectually and personally, and you truly inspire me.
- My family for all of their love and support. My sisters for listening to me ramble on about engineering problems. My Mom for always encouraging me, and for helping me with some difficult decisions. My Dad, for being the best role model I could ask for, and for sharing his love of this business with me. I'm very proud to be your daughter.
- And finally, Al, for all of his constant love, friendship, and support. I am the luckiest girl in the world. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

As technology progresses, systems are growing in both capability and complexity. This trend is true for general technological systems, including but not limited to aerospace systems. Systems are becoming more multi-functional, involving the interaction and interfacing of many sub-systems and, often times, many engineering disciplines. For example, distributed computer networks, linking computer and processing engineering to communications engineering, are many times more powerful than the single desktop computer of only a few years ago. While this increased capability allows the systems of today, and the future, to last longer and perform multiple, more difficult tasks at once, the increased complexity makes these systems more difficult to design. Thus, a designer can no longer use simple engineering judgement to directly see which components are crucial to the system and should therefore be focused on during the design process. The effect of a failure in one sub-system on the system as a whole is no longer clear, and neither is the effect of improving one sub-system or component. The pure magnitude of the number of components and interactions in these systems has made it nearly impossible to understand how the system will behave without modeling.

Once models are developed for these systems, it is possible to optimize the design based on the reliability of the system. The reliability of the system can be increased in one of two ways: increasing redundancy or increasing the reliability of the components. However, reliability is not the only life cycle metric of interest to designers. Other metrics of

interest include life cycle cost and productivity. The particular combination of redundancy and improved reliability of components used in a system design will effect not only the system reliability, but these other life cycle metrics as well. Therefore, this research focuses on the question of where funds should be spent to either add redundancy or improve the reliability of components, in order to achieve the best combination of life cycle metrics possible.

1.1 Motivation

Future space missions are becoming increasingly complex and larger in scale, and are also becoming more difficult to accomplish with single spacecraft. This has led many programs to examine the use of Distributed Satellite Systems (DSS). These systems use smaller, multiple spacecraft to get the same benefits as one larger spacecraft [Shaw, Miller, and Hastings, 2000]. In addition to enabling complex and large systems, distributed satellite systems also offer several other benefits. The spacecraft in a DSS cluster can be smaller and less complex than single spacecraft counterparts. This can lead to shorter development times and, in turn, reduced life cycle cost. Due to the modular design and decentralized resources, these systems are both survivable and upgradable. Staged deployment of a distributed satellite system can be beneficial in several ways. First, individual satellites with technology readily available can be launched sooner, instead of waiting for the technology needed to accomplish the entire mission to be ready. While in this stage the system may not be complete, some scientific data may be able to be collected. By using staged deployment, programs can also spread out both cost and risk. In addition, older spacecraft can be interchanged individually with spacecraft containing new technology. Lastly, if one spacecraft were to fail, that individual spacecraft could simply be replaced without having to replace the entire cluster.

One of the areas in which distributed satellite systems could prove the most useful is space-borne interferometers. NASA's stated goals for future missions can no longer be accomplished with single aperture telescopes. In particular, NASA's Origins Program has

the goal of finding, characterizing, and studying Earth-like planets around distant stars. To accomplish the angular resolution required to achieve such goals, 100-meter or larger single-aperture telescopes would be required. To study these distant planets the light from the nearby star will need to be blocked out. One possibility for accomplishing this is to use nulling interferometry. Another powerful tool for future missions would be interferometers with variable baselines. Interferometers with smaller baselines are better for resolving individual large targets, while interferometers with larger baselines are better for resolving between two small targets. Therefore, an interferometer that could vary the baseline depending on the observation and situation would be an extremely powerful tool. One method of achieving a variable baseline interferometer is to put apertures on individual satellites and use a distributed satellite system. This type of interferometer is referred to as a separated spacecraft interferometer and is being considered for several future missions.

Jet Propulsion Laboratory's (JPL) Terrestrial Planet Finder (TPF) will survey near-by stars to detect, image, and characterize any Earth sized planets with atmospheres [JPL, TPF, 2001]. TPF will use spectroscopy to characterize the atmospheres of the planets it finds, to determine if there are any planets near Earth that have a high potential for life. This mission is the next step in the exploration of the universe, and is considering using a mid-IR separated spacecraft interferometer to achieve these goals. The Micro-Arcsecond X-ray Imaging Mission (MAXIM) Pathfinder will lead the way for the MAXIM mission, which will resolve the event horizon of black holes [GSFC, MAXIM, 2001]. This has several scientific implications including imaging black holes, testing the theory of relativity, and learning more about gravity. To achieve these goals, MAXIM Pathfinder will need an instrument with an angular resolution of 100 micro-arcseconds, and will use an X-ray separated spacecraft interferometer. Life Finder is the next mission in the ORIGINS program, after TPF, and will be more sensitive version of it's predecessor [JPL, LF, 2001]. Life Finder also does spectroscopy on the atmosphere, but this time looks for dips in energy, which is a sign of life. A separated spacecraft interferometer is being considered for this mission which will provide the first direct detection of life on other planets. The

Submillimeter Probe of the Evolution of Cosmic Structure (SPECS) will attempt to answer the fundamental questions of how the universe began and how it evolved to where it is today [GSFC, SPECS, 2001]. It will accomplish this by observing the first and most remote galaxies of the universe and taking measurements of protostars and developing planetary systems in the Milky Way or nearby galaxies. A far-IR interferometer is proposed to take these measurements on either separated or tethered spacecraft. The Stellar Interferometer will provide the best forecasting of solar activity and study the impact of stellar magnetic activity on astrobiology and life in the universe [GSFC, SI, 2001]. Specifically, the Stellar Interferometer will study the various effects of magnetic fields of stars, what generates them, and the internal structure and dynamics of the stars in which they exist. This new knowledge will help scientists better predict climate and solar flares which can effect communications satellites. The Stellar Interferometer will be a UV interferometer, possibly on separated spacecraft. The Laser Interferometer Space Antenna (LISA) will attempt to observe and prove the existence of gravitational waves [JPL, LISA, 2001]. The proof of these waves existence would unify many scientific theories and provide one of the fundamental building blocks of the current theoretical picture of the universe. LISA will also use a separated spacecraft interferometer to accomplish its mission goals.

While separated spacecraft interferometers are one of the only tools powerful enough to achieve some of NASA's future goals, they also involve several technology areas which are not currently fully developed, including high precision formation flying and spaceborne interferometry. At the present time, separated spacecraft interferometers are both very expensive and very risky. It has been noted that approximately 80-90% of the development cost of a large system is predetermined by the time only 5-10% of the development time has been completed [INCOSE, 1998]. This is because early decisions on the architecture of a system affect almost all aspects of the system further down in the design process. Therefore, these high-level systems architecture decisions need to be examined carefully and based on quantitative models.

The research discussed here examines how to design a top-level separated spacecraft interferometer system in order to maximize life cycle throughput. Although the research described is specifically related to separated spacecraft interferometers, several of the algorithms and tools to be developed are applicable to space systems in general.

1.2 Problem Statement

Neither formation flight nor space-borne interferometry, the two main technologies behind separated spacecraft interferometry (SSI), is a fully developed nor proven technology. This implies two consequences that affect SSI systems: extra cost and extra risk. The extra cost comes from the need to develop these technologies to an operational level before they can be used in a NASA mission. The extra risk inherent in SSI systems stems from the fact that they involve new, complex systems and processes. This combination of extra cost and extra risk leads to a need to develop these systems with the concept of maximizing the overall productivity, or throughput, of the system at the given cost. Maximizing the throughput of a system involves two separate processes. First, it is important to maximize the reliability of the system so as to minimize the probability of a failure occurring. Second, with an inherently risky system it is important to realize that even with the maximum system reliability, the probability of a failure is higher than in a non-risky system and needs to be planned for. Therefore, it is important to maximize the productivity of the system in the event that failures do occur in order to ensure the reported performance is the performance of the system over the entire lifetime and not simply an instantaneous performance.

Reliability is defined as the probability that a system will be in a functional state at the end of its given lifetime. When billions of dollars and many years are invested in a project, it is extremely important to ensure that the system has as high a reliability as possible. There are two distinct ways to increase a system's reliability. The first is to increase the reliability of the individual components or sub-systems. A system's total reliability is the product of all component reliabilities if the components work in series and therefore all compo-

nents are required for the system to remain functional. Research and extensive testing can improve component and subsystems reliabilities. The second method of improving reliability is to increase redundancy. With redundancy if one component fails, a second component can still perform the operation required. This leads to a robust design since the system can sustain failures and still function. With complex systems, such as interferometers, a combination of increased component reliability and redundancy is required to create the most reliable system possible. The weighting and distribution of this combination is not clear, however. How much money should be spent improving component reliability versus buying more components to implement redundancy? Which component's reliabilities should be improved and which components should be duplicated? These decisions are not obvious. Models that calculate the reliability of a given architecture, along with failure analyses and optimization programs can help to answer these questions.

While it is important to design a system such that the highest reliability possible is achieved, it is also important when dealing with systems using new technologies to design for situations in which failures have occurred. In other words, it is important to not only design a system that will continue to function in the event of a failure, but will also function at as high a productivity rate as possible. To accomplish this, it is necessary to model the system and look at the productivity in each possible state. A state is defined as the functional system that remains in the event of failures. Modeling the productivity in each state requires two aspects: a method to model the productivity of a system given the system parameters, and a Markov, or state-transition, model of the initial system. To model the productivity of a system, a basic understanding of the system is required. In this case, the system being modeled is an interferometer. The model of the productivity of an interferometer can vary from very basic equations depending only on the number of collecting apertures, to very complex equations involving many more system parameters. The research described here uses a relatively low fidelity model, described in Section 2.2 and Chapter 3. The Markov model analyzes transition rates from one state to another state and can lead to the probability that the system is any given state at any given time. This model assumes that the probability of being in the initial, or no-failure state, at the beginning of

the simulation is one, and that the system transitions from one state to another at rates equal to the failure rates of each component. With the probability of being in each state throughout time known, and with a model to estimate the productivity of the system in each state, a total estimated productivity, in the event of failures, can be calculated. Once this estimated productivity is known, architectures can be compared in terms of total estimated productivity.

While both reliability and productivity are important aspects to any system, the architecture with the highest reliability does not necessarily have the highest productivity and vice versa. The public will not be happy if the government spends money to buy a system which has a very high reliability, and will therefore last for decades, but will produce one image every year. Conversely, the public would also not accept a system which produced an image per minute but lasted only one day. For this reason, it is important to design a system which has an acceptable level of both reliability and productivity. Additionally, other parameters, such as total system cost, play a key role in determining the usefulness of a system and success of a program. As an example, a program which develops a system that has a very high reliability and productivity but costs double the allowable budget will not get the funding to get beyond the first planning stages. Therefore, a method of finding an architecture that gives the best combination of all relevant parameters needs to be found. System models - including reliability, productivity, and cost models - and optimization programs can be used together to arrive at such an architecture.

1.3 Research Objective and Expected Results

The objective of this research is to develop models and comparison and optimization programs to help answer the high-level architecture design questions discussed above. This effort is comprised of both interferometry systems research and optimization methods research. Specifically, this research effort aims to:

1. Develop Markov model analysis tools to evaluate the probability that a system is in any given state at any time throughout a mission lifetime. Define states of the system based on knowledge of operation and failure analysis.
2. Create productivity model of interferometry systems. Use model and Markov analysis to estimate total productivity of individual systems.
3. Create cost model and estimate cost of individual interferometry systems.
4. Use Markov model analysis to estimate reliability of individual systems. This step also involves creating a model to estimate how much a component's reliability is improved for a given amount of money spent.
5. Use all information described above to compare multiple user-defined architectures in terms of total performance.
6. Use optimization tools to optimize architecture level design decisions to find the optimal method of dividing money between increasing reliability of components and additional redundancy to achieve the best combination of life cycle metrics possible.
7. Develop sensitivity analysis of these results to unknown inputs, such as failure rates and cost model parameters.
8. Create Matlab toolbox to run all analyses discussed above. The toolbox should be able to both compare user-given architectures and optimize over the entire design-space to allow for use in multiple stages of design.

The results of this effort will be applied directly to the conceptual design of separated spacecraft interferometers. In addition, the tools created will be applicable to all space systems which are complex and consist of multiple components.

The expected results of this research are rules of thumb and tools to be used to determine high-level architecture design decisions for interferometer systems. These decisions include which components require redundancy and which components should be developed more fully to achieve a higher individual reliability. Other decisions include how many of each component, as well as type of spacecraft (combining, collecting, and dual functioning), create the optimum combination of high reliability, high productivity, and low cost for given program goals. In addition, the sensitivity of all of these analyses to unknown parameters will be estimated.

Chapter 2

SEPARATED SPACECRAFT INTERFEROMETRY

While the question of how to spend funds to improve reliability, productivity, and cost by either adding redundancy or increasing the reliability of components is applicable to all complex systems, the research presented here focuses on one type of system in particular: separated spacecraft interferometers. Separated spacecraft interferometers are being considered for use in several future NASA missions, as discussed in Chapter 1. While the models used throughout this research are low fidelity, a basic understanding of the system being modeled is still required to understand the model itself. Therefore, this chapter will provide the reader with a basic understanding of how interferometry works. It should be noted that this chapter is by no means meant to fully explain interferometry systems, but simply provide enough information to allow the reader to understand the models used throughout this research. Following this introduction, this chapter will then familiarize the reader with the model of these systems used later in this research.

2.1 Separated Spacecraft Interferometry Background[1]

Resolution is one of the key parameters that is used to describe the performance of a telescope. A telescope with a small angular resolution can make out smaller objects than one with a large angular resolution. This parameter improves with increased diameter of the

1. This section is based on conversations between the author and both Oliver Lay, of JPL, and David Miller, of MIT [Lay, 2001;Miller, 2001].

main aperture, or mirror. As mirrors get larger however, they get more impractical to launch into space. An interferometer is a type of telescope that uses multiple smaller mirrors instead of one large one. With this method, if two one-meter diameter mirrors are placed a kilometer apart, they will have the same angular resolution as a one-kilometer diameter mirror. This method of improving resolution is very powerful for space-based telescopes, since launch costs can be dramatically reduced.

A telescope creates images by collecting photons from the target that it is observing. In celestial observations, the targets are usually stars. As the distance to a star increases, it can eventually be considered a point of light with no angular diameter. A typical optical telescope with a single main aperture collects photons in the manner shown in Figure 2.1. Please note that throughout this discussion, apertures are assumed to be square, and not circular as is most often the case. The concepts are easier to visualize this way and were therefore used in this introductory discussion. If a circular aperture is assumed however, the distance labeled as λ/d in Figure 2.1 would actually be approximately 1.22 times λ/d since there is more light coming from the exact center of the target than there is from the sides. This concept can be seen in Figure 2.2.

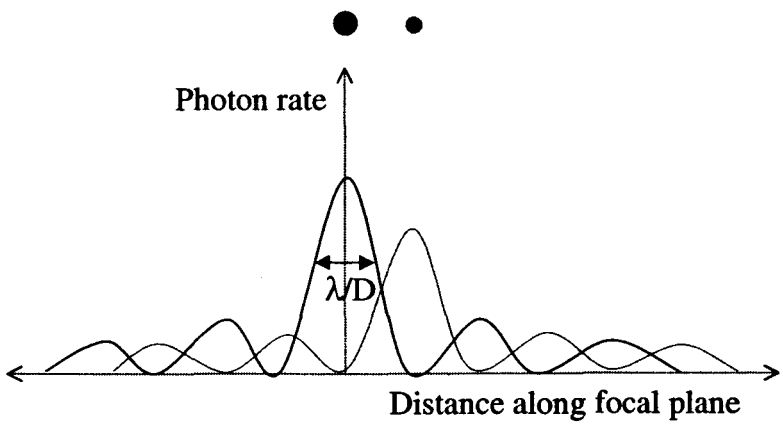


Figure 2.1 Photon rates for a single aperture optical telescope

In Figure 2.1, the black line is the photon rate from a single point of light, represented as the black dot. If a second point of light is next to the initial point of light, as in the green

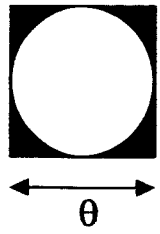


Figure 2.2 Difference between a circular aperture and a square aperture

dot, it will create a similar pattern, slightly shifted, as seen by the green line. In this case the second, or green, point of light is fainter than the first, or black, point of light. This can be seen from the relative amplitudes of the peaks. The actual data from the telescope would not show these individual patterns, but the sum of the individual patterns. The angular resolution of a typical optical telescope is given by Equation 2.1.

$$AngularResolution = \frac{\lambda}{D} \tag{2.1}$$

In Equation 2.1, λ is the wavelength of the light being observed, and D is the diameter of the main aperture. If the second point is too close to the first point, then the peaks will overlap and the telescope will not be able to see a distinction between the points. The second point, or star, needs to be separated from the first point, or star, by the width of the peaks in order for the stars to be distinguishable. This gives the equation for angular resolution.

Interferometers are similar to single aperture optical telescopes in many ways. The images are again created by collecting photons from the targets. With interferometers, a number of measurements are required to get a complete image of a target. The number of measurements required depends on the complexity of the target. Only a few measurements of each target, no matter what the complexity is, are required to get useful information about the target, however. An example of the data an interferometer would measure can be seen in Figure 2.3a. This pattern of photon rate versus projected angle in the sky is called a fringe, and is used to determine information about the target. The distance

between the two collecting apertures is known as the baseline and is represented as B. The pattern in Figure 2.3a is similar to that from an optical telescope in Figure 2.1, in that the peaks are of approximately equivalent widths, with the diameter simply replaced by the baseline. The main difference is that the peaks in the pattern from the interferometer do not decay, but rather are of constant amplitude. The effect of the baseline can be seen in Figure 2.3b. As the baseline increases, the frequency of the fringe increases. The amplitude of the peaks is again dependent on the magnitude of the target.

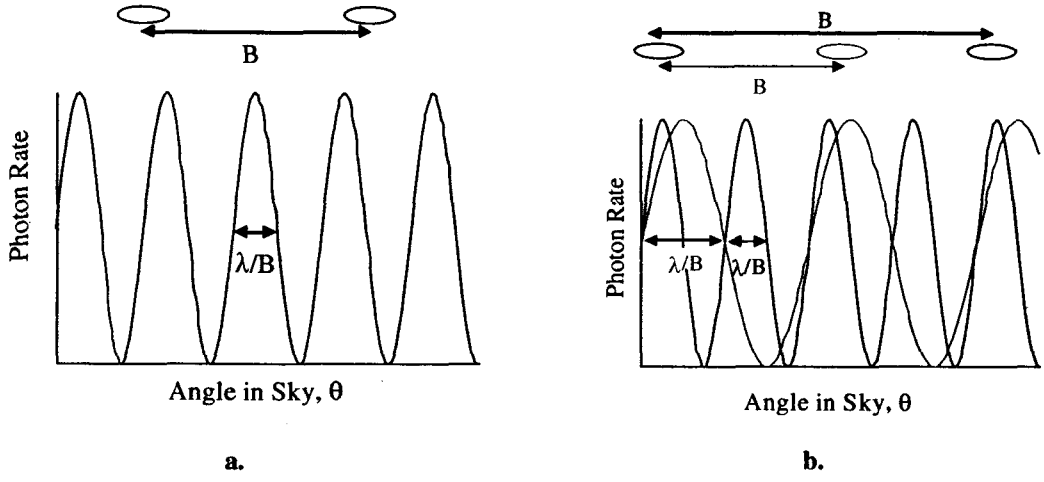


Figure 2.3 a) Photon rates for a single baseline interferometer. b) Photon rates for an interferometer showing the effects of varying baselines

There are two ways to think about the axes of a fringe. If a fringe is considered the pattern a target would make in the focal plane, the x-axis is the amount of offset in the delay line, or the optical path difference (OPD), and the y-axis is the photon rate. A fringe can also be thought of as a projected pattern on the sky. In this method the light from a target travels the same distance to both apertures when it is directly in line with the middle point of the two sets of optics, giving rise to a peak. As the star moves to one side or the other of the middle point, the light begins to travel slightly different distances to each aperture, causing varying constructive and destructive interference levels, and giving rise to a fringe pattern. In this case, the x-axis of the fringe is the angle in the sky of the target compared to the mid-point of the apertures and the y-axis remains the photon rate. Figure 2.3 shows

this second method of portraying a fringe, while Figure 2.4 shows an example of this angular offset. If the first method of portraying a fringe is used, the pattern remains the same as in Figure 2.3, however the axes change and the width of the peaks is no longer λ / B, but rather the distance between the peaks is equal to λ, as shown in Figure 2.5. The patterns shown in Figure 2.3 and Figure 2.5 are representative of a single point in the sky with zero angular width.

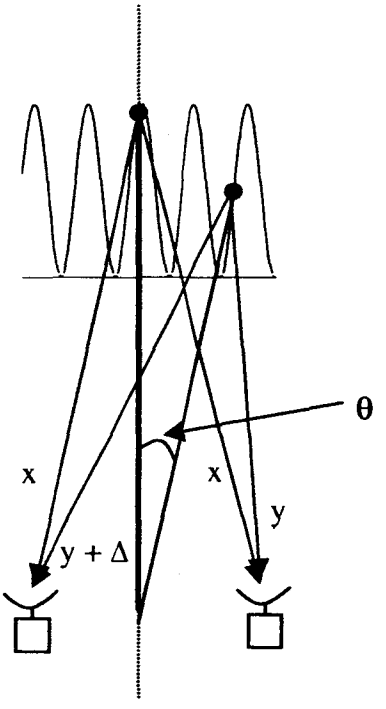


Figure 2.4 Definition of angle in the sky.

2.1.1 Visibility

Unless a star is infinitely far away, it is actually wider than a single point in the sky. This width is effectively seen in interferometer measurements as individual points of light next to each other that together are as wide as the actual star. This can be seen in Figure 2.6.

In Figure 2.6, each point is separated by an angular distance θ. Note that the different colors in Figure 2.6 represent different points of light, all with the same wavelength. Light

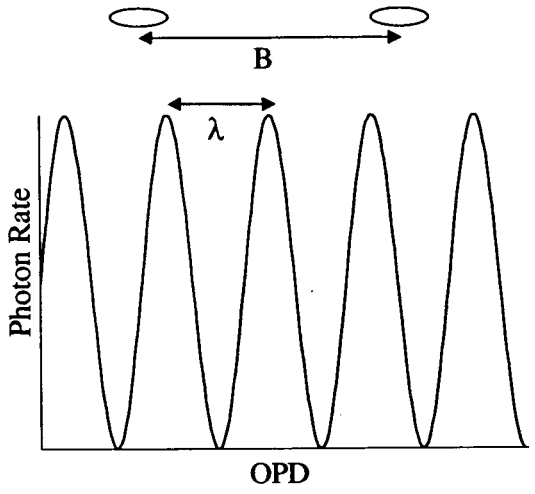


Figure 2.5 Photon rates for an interferometer, seen as the pattern a target would make in the focal plane.

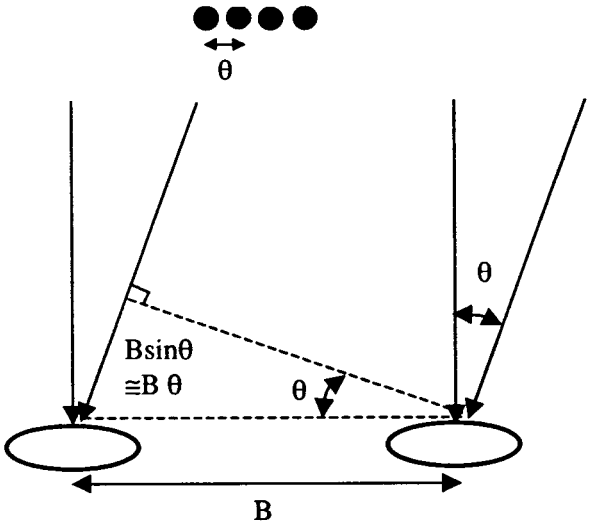


Figure 2.6 Collecting light from multiple points in an interferometer

coming straight into both apertures is compared to light coming in at an angle θ, implying the target is offset in the sky by this same angle. This offset causes the light to travel *Bsin*θ further to one aperture than to the other. This extra distance can be approximated as *B*θ for small angles. The effect on the fringe pattern is similar to the effect of adding a second point to the pattern in an optical telescope. This effect can be seen in Figure 2.7.

A target represented by the four points shown in Figure 2.6 would be 4θ wide. As the star gets wider and wider, there are more and more points of light next to each other. As with the optical telescope, these individual patterns for each point do not appear in the data, but rather the sum of all photons is recorded. The sum of the individual patterns can be seen in Figure 2.9 for a single point of light up to four points of light. Note that as more points of light are summed, implying a larger star, the total photon rate becomes more constant, with less relative difference between the valleys and the peaks. It is this difference that is used to measure the angular size of the star in the sky.

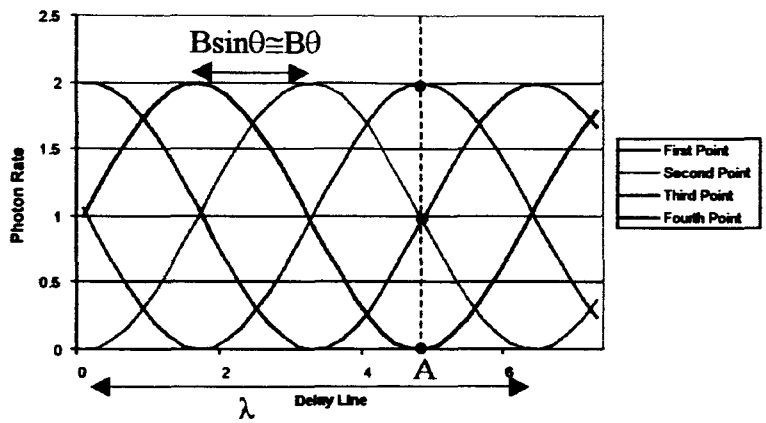


Figure 2.7 Photon rates from multiple points of light

Visibility is a parameter used to measure the relationship between peaks and valleys of a fringe. Figure 2.8 and Equation 2.2 illustrate the method of calculating fringe visibility.

$$vis = \frac{A}{B} = \frac{0.5(x-y)}{0.5(x+y)} \tag{2.2}$$

If a single point of light is measured, as in the black line in Figure 2.9, the valleys of the pattern have zero amplitude. Therefore, the y variable in Equation 2.2 is zero, and the visibility is equal to one. However, if two or more points are seen together, as in a star with angular width, the valleys no longer have amplitudes of exactly zero. For example, in Figure 2.7 and Figure 2.9 the point A has zero photon rate if just the black line is mea-

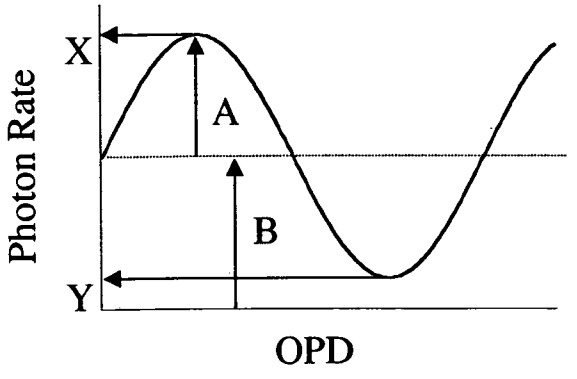


Figure 2.8 Visibility calculation definitions

sured. If multiple lines are summed, as is the case with the green, blue, and red lines in Figure 2.9, the sum is no longer zero. These summed lines in fact will never reach a value of zero. Therefore the value of y in the equation for visibility will no longer be zero, and the visibility will no longer be one.

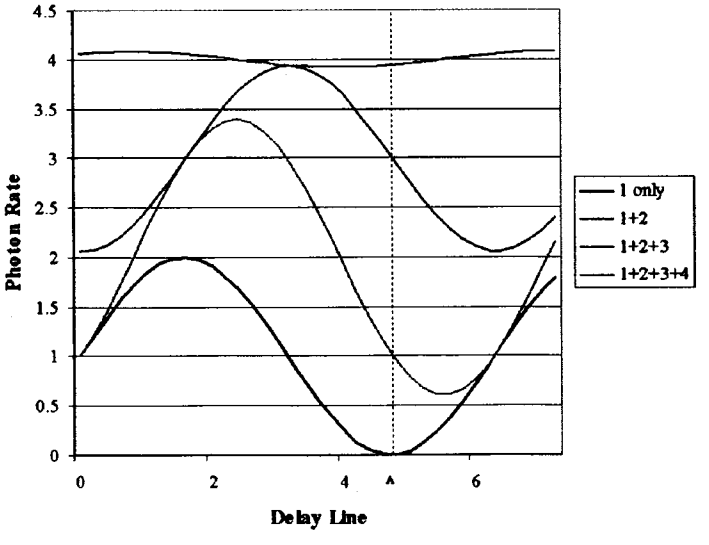


Figure 2.9 Photon rates from multiple points of light

As more patterns are summed the relative sum of the peak and the valley, B, increases, while the relative difference, A, decreases. This causes the visibility to decrease. In the

extreme case, if the fringe were a constant value, x would equal y in Equation 2.2, and the visibility would be zero. Therefore a single point of light will have a visibility of exactly one, while a star of infinite width will have a visibility of zero. This relationship can be seen in Figure 2.10.

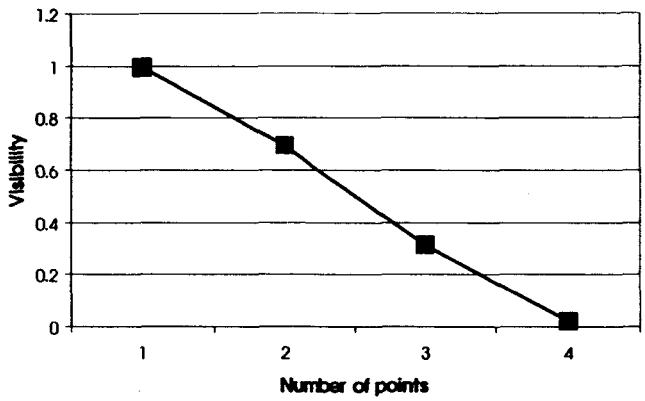


Figure 2.10 Visibility comparison for up to 4 points

As shown in Figure 2.7, the distance between the patterns of individual points of light goes approximately as the baseline times the distance between the points. Therefore as the baseline decreases, so does the distance between patterns in Figure 2.7. If this distance becomes too small, then it becomes impossible to distinguish between the peaks. This is similar to angular resolution of the optical telescope. If these patterns are too close the visibility will not drop, since the pattern of the individual points summed up still reaches near zero. This effect can be seen in Figure 2.11 a, b, and c. Compare Figure 2.11 to Figures 2.7, 2.9, and 2.10 to see the difference between a large $B\theta$ and a small $B\theta$.

Assuming an ideal instrument, visibility is always equal to one for a single point of light and for a star of any width if the baseline is very small. As the baseline increases, visibility drops. The sharper the drop in visibility, the larger the angular width of the star. As the baseline increases past the first minimum in visibility, visibility begins to increase again. This effect could be seen by adding more points to the example shown in Figures 2.7, 2.9,

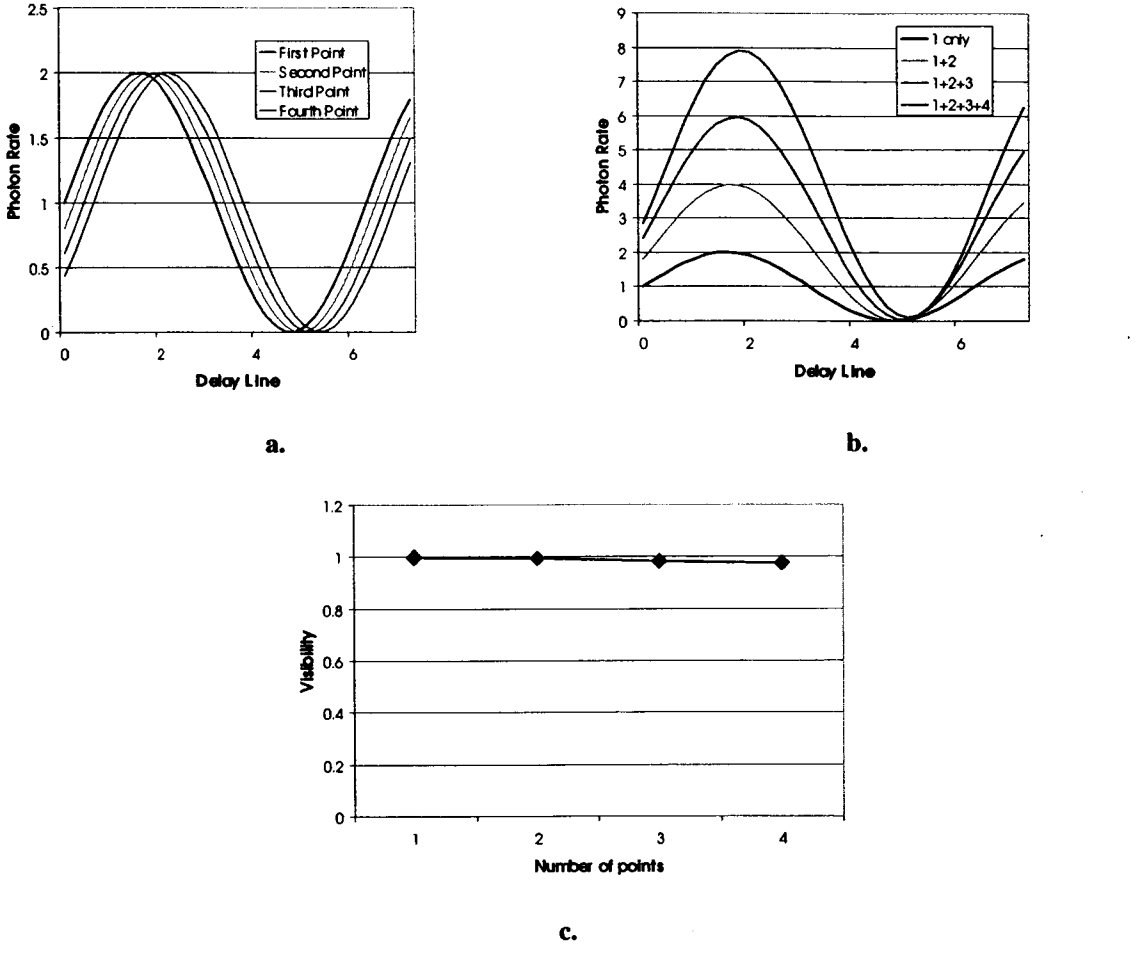


Figure 2.11 4 points of light separated by 0.2 units (Figures 2.7, 2.9, and 2.10 are separated by 1.6 units). a) Photon rates. b) Total photon rates. c) Visibility comparison

and 2.10. A fifth point would add a second line basically on top of the black line in Figure 2.5 and double the component from the first, or black, point in the sum. This would cause a larger visibility than if each component is only counted once, as shown. These later peaks in visibility are much smaller than the first peak however, and will eventually taper out to zero. This can be seen in Figure 2.12.

Figure 2.12 is simply an example of the trends of visibility in relation to angular width of the target and baseline of the interferometer. Curves similar to those in Figure 2.12 exist with the exact relationship between these three parameters. Therefore, the size, or angular width, of a star, if it is assumed to be circular, can be measured by simply measuring the

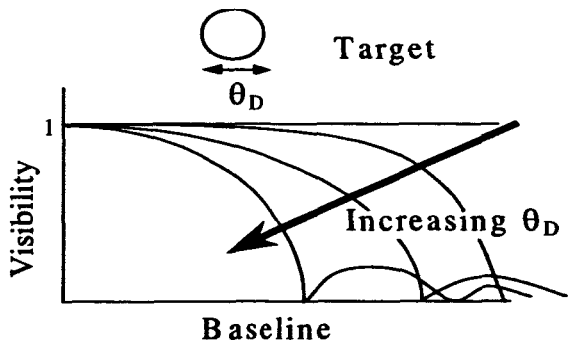


Figure 2.12 Relationship between visibility, baseline, and target size

visibility of the target at one baseline, which is assumed to be known. The measurement can then be fit to one of the pre-existing curves to determine the size of the target.

Visibility can be measured by measuring the photon rate, N, at any four points along a wavelength in the fringe. This can be seen in Figure 2.13 and Equation 2.3.

$$vis^2 = \frac{(N_A - N_C)^2 + (N_B - N_C)^2}{(N_A + N_B + N_C + N_D)^2} \tag{2.3}$$

Figure 2.13 shows points *A, B, C,* and *D* falling directly on or half way between the exact peaks and valleys of the fringe. This does not need to be the case for Equation 2.3 to hold. The point *A* can be anywhere along the fringe. The points *B, C,* and *D* simply need to be measured relative to the point *A*, at exactly one-quarter wavelength intervals. Therefore, once the fringe is found for a given target, the visibility can be calculated from four measurements along that fringe.

2.1.2 Resolution

As mentioned before, a star or target with angular width can be thought of as individual points of light next to each other for a distance equal to the angular width of the target. If enough of these points are next to each other, the entire area within the pattern shown in Figure 2.7 will be covered. This can be seen in Figure 2.14. If the pattern is entirely filled in, there is no way to make out even a small fringe in the total photon rate, which

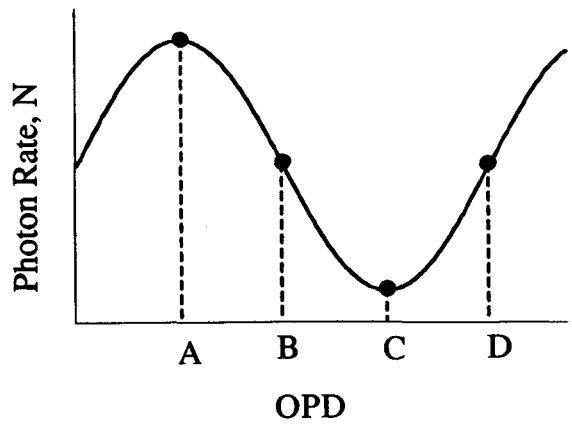


Figure 2.13 Visibility calculation definitions.

approaches a constant value. This can be seen in Figure 2.14b. If the total photon rate is constant, then the visibility is exactly zero, and the interferometer cannot resolve anything. This is known as resolving a star out. This occurs if the baseline times the angular width of the star (i.e. the distance between the first pattern's peak and the last pattern's peak - see Figure 2.7) is much greater than the wavelength of the light being observed. Therefore it is difficult for an interferometer at a given baseline, B, to resolve a star that has a larger angular width than the wavelength of the light divided by the baseline. If the angular width of a target is larger than this angle, the interferometer can only decipher that the target is larger than its capability to see, but cannot decipher any information on how much larger the target is. The angular width at which a star is resolved out, θ_{RO}, is given in Equation 2.4. Note that in order to resolve individual targets of large angular width, the baseline should be small.

$$\theta_{RO} \gg \frac{\lambda}{B} \tag{2.4}$$

There remains the question of how close together two individual targets can be for an interferometer to be able to distinguish between them. For example, how close can the two stars, or point sources, in a binary system be before the interferometer sees them as a single point source? A binary system would create a visibility pattern similar to that

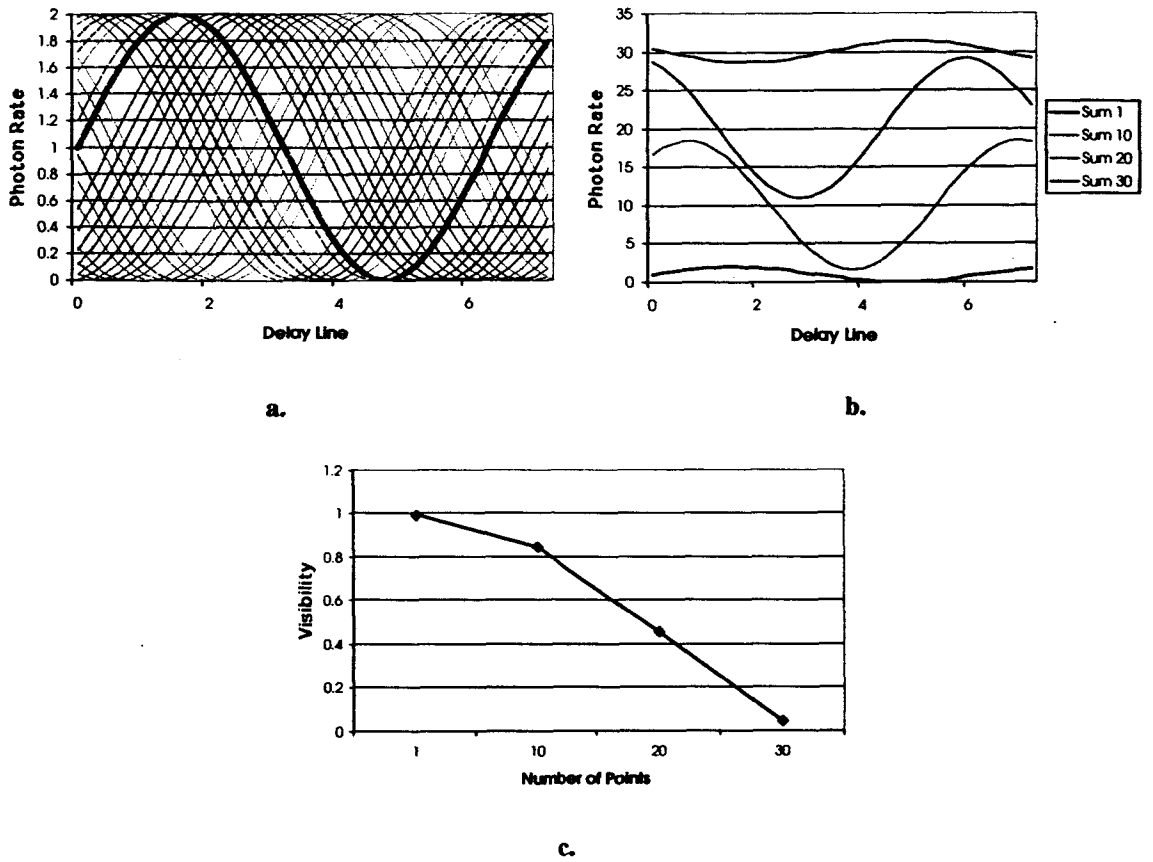


Figure 2.14 Example of a resolved out star - 30 points separated by 0.2 units. a) Photon rates. b) Total Photon rates. c) Visibility comparison

shown in Figure 2.15. The first star would create the usual pattern shown in Figure 2.3a. The second star would create the same pattern, shifted over by an amount equal to the baseline times the angular separation between the two sources. When this shift is exactly equal to one-half the wavelength of light, the two patterns will add together and cause complete destructive interference. In other words, the total photon rate would be constant, causing a visibility of zero. This accounts for the null at lambda over two times the angular separation in Figure 2.15. As the separation between the patterns continues, this process continues. Eventually, when the baseline times the separation of the sources is equal to exactly one wavelength, the two patterns have complete constructive interference, and the visibility is once again one. To determine that there are two sources and not simply

one larger source, the baseline must be able go past this null and see the second maximum peak in Figure 2.15.

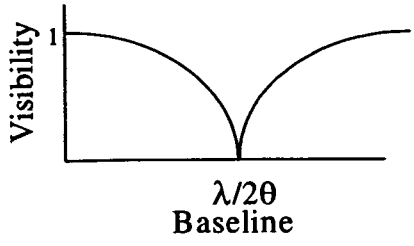


Figure 2.15 Visibility for a binary system.

It is also possible to look at the angular resolution of an interferometer in a similar manner to the discussion of the angular resolution of a single aperture optical telescope. If the patterns in Figure 2.7 are too close together, as in Figure 2.11, then they cannot be distinguished from one another. One criterion for when peaks can be distinguished is that the peak of the second pattern cannot be within the distance from the peak to the trough of the first pattern. For a square aperture, in which all the light has the same distance to travel, the first trough is at one half the wavelength. This is the easiest concept to visualize and has therefore been used in all previous discussions. For a circular aperture, the first trough is actually at approximately 1.22 times the wavelength, since there is more light coming from the exact center of the target than from the sides. This can be seen in Figure 2.2.

All of these arguments put the smallest angular separation at which an interferometer at a given baseline can still make out two individual targets, θ_{RES}, between the wavelength over the baseline and the wavelength over two times the baseline. This can be seen in Equation 5. The range in this equation is due to the fact that there is no strict value of where the instrument can specifically separate two targets. The area in which it can and cannot separate targets blends together smoothly, and where exactly the cut-off is can be unclear or indistinguishable.

$$\frac{\lambda}{2B} \leq \theta_{RES} < \frac{\lambda}{B} \tag{2.5}$$

It is worth noting that in order to resolve individual large targets (large θ_{RO}) a small baseline is needed (see Equation 2.4), but in order to resolve between two close small targets (small θ_{RES}) a large baseline is needed (see Equation 2.5). With a large baseline, details of an image can be resolved, but the background and large areas in the image would be resolved out. With a small baseline, the large areas and backgrounds can be resolved, but no detail would come through. This is one reason that any interferometer attempting to image a target needs a variable baseline. This is one of the main arguments for using a separated spacecraft interferometer, because it allows for a totally variable baseline.

2.1.3 Imaging

In addition to the size of a target, an interferometer can also be used to gather information about the shape of a target. Figure 2.16 illustrates this process. If a target is actually an ellipse, rather than a circle, then the size information given by an interferometer with collecting mirrors horizontally across from one another would be different from the size information given by the same interferometer with the same baseline, but with the collecting mirrors vertically across from one another. This difference in measurements implies that the target is elliptical in shape. Therefore, an interferometer can begin gathering data on the shape of a target with just two measurements. These two measurements are represented by the red dots in Figure 2.17. The plane in Figure 2.17 is known as the UV-plane and each measurement taken at a given baseline at a given orientation produces two UV points. If the entire UV-plane is filled in within a circle with a radius of the largest baseline used, a fully sampled image can be created. The transformation from the UV-plane to the image plane is accomplished through a Fourier transform. With two points, the size can be determined. With four points, similar to the four red points in Figure 2.17, the basic shape can begin to be determined. With the entire plane filled in uniformly, as in the black dots in Figure 2.17, the entire shape of the target can be determined, and an image can be taken. The change in angle around the circle is used to gather shape information, and the change in radius through the circle is used to gather both detailed and large area information, as was discussed above [Lay, 2001].

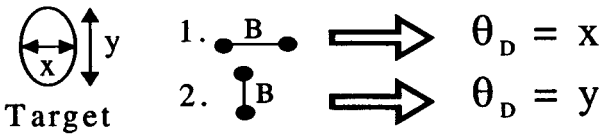


Figure 2.16 Resolving a target's shape

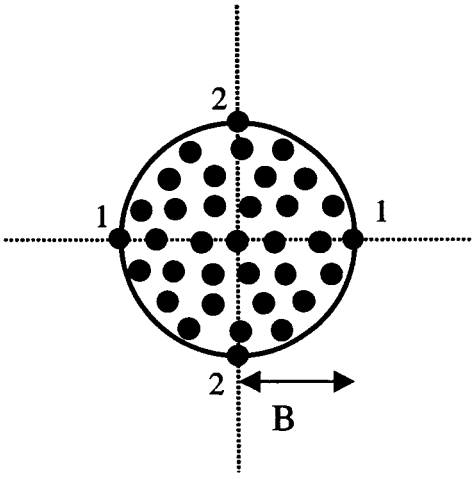


Figure 2.17 Sample UV-plane

2.1.4 Broadband Light

The previous discussion has involved a simplification to assist in the visualization of concepts. The light discussed above is assumed to be monochromatic, or single wavelength. While this simplification makes interference and other concepts much simpler to visualize, it is almost never physically realizable or useful. In reality, most light being studied by an interferometer has components of different wavelengths. When this occurs, it is impossible to get a visibility of exactly one. This is due to the fact that even if one component were shifted by exactly one wavelength, that shift would not be exactly one wavelength for a different component. In other words, the instance in which the optical path difference is exactly zero is the only instance in which all the light from both sides lines up exactly. At any other optical path difference other than zero, the light from at least one wavelength component will not be lined up exactly from the two apertures. This implies that the pattern the interferometer will receive for broadband light will have a maximum

when the light is completely constructively interfered. The general pattern will remain the same as with monochromatic light as the OPD is increased and decreased in the sense that it will still vary between peaks and valleys. However, the peaks will continuously decrease in magnitude while the valleys will never be zero and will continuously increase in value. This pattern can be seen in Figure 2.18. The visibility, as defined in the previous discussion, of broadband light is measured at the center of the fringe using the first (zero-point) peak and valley amplitudes.

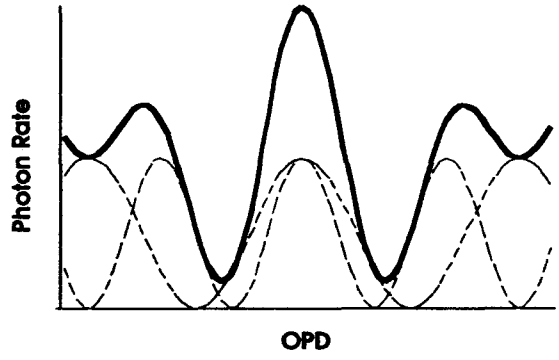


Figure 2.18 Photon rates for broadband light. The red and blue lines are individual wavelength components and the black line is the sum.

Figure 2.18 implies that the theoretical limit on visibility for broadband light is not one, but a value lower than one since the first valley will never be zero. The specific theoretical limit is different for different combinations of wavelengths. For example, white light is comprised of a component of every wavelength. The photon pattern an interferometer would record for a point of pure white light would then be an impulse at zero OPD, with an amplitude dependant on the magnitude of the light being observed. At any point other than zero OPD the photon rate would average to a constant, at an amplitude of one half the amplitude of the impulse. This pattern can be seen in Figure 2.19, and is simply an extreme case of the pattern shown in Figure 2.18. If these numbers are plugged into Equation 2.2, the theoretical limit on visibility for white light is shown to be one-third. This calculation is shown in Equation 6.

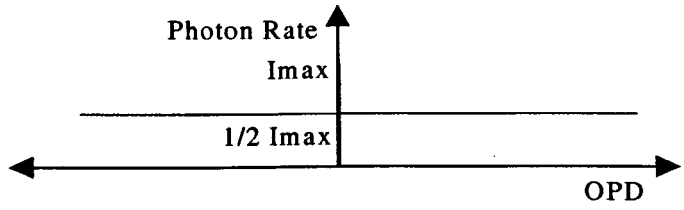


Figure 2.19 Photon rate for white light.

$$MaxVisibility = \frac{(1/2)(I_{max} - (1/2)I_{max})}{(1/2)(I_{max} + (1/2)I_{max})} = \frac{1/2}{3/2} = \frac{1}{3} \tag{2.6}$$

Even when observing monochromatic light, a physical interferometer would never read the theoretical maximum visibility of one. This is due in part to vibrations. As vibrations move through the interferometer, the point on the delay line at which the light from both apertures travels exactly the same distance, and therefore the fringe would in effect vibrate as well. This will cause even a single point source to appear similar to the pattern shown in Figure 2.11, which in turn will cause the point source to have a very high visibility reading, but it will not be exactly one. As more vibrations are damped out of the instrument, the maximum visibility for monochromatic light through that instrument will asymptotically approach one, but the limit itself is purely theoretical [Miller, 2001].

2.2 Separated Spacecraft Interferometry Model

The major complication when operating an interferometer is that the light that comes from one end of the instrument needs to travel exactly the same distance to the combining optics as the light coming from the other end of the instrument in order to get a fringe. Therefore, if all the spacecraft that are capable of collecting light in a system are placed symmetrically about some center point, and a spacecraft capable of combining light is placed at this center point, the system should be able to collect fringes. Each fringe is equivalent to one visibility reading, and therefore accounts for two UV points. Each set of spacecraft capable of taking a measurement for two UV points forms what is known as a baseline. In this sense, an equal number of baselines and sets of UV points are needed to

complete an image. The more spacecraft collecting light in each configuration, the fewer times a change in the configuration of the system is needed to get more baselines, and therefore fill in the UV-plane. While only one set is needed at a time, an interferometer system can not function without at least one set of combining optics. For this reason, any model of a separated spacecraft interferometry system needs to capture the number of collecting and combining apertures available.

Throughout this work, each separated spacecraft interferometer system was modeled as a combination of three different individual types of spacecraft. The three types of spacecraft available are combining spacecraft, collecting spacecraft, and dual functioning spacecraft. As the names suggest, each collecting and dual functioning spacecraft is capable of collecting light while each combining and dual functioning spacecraft is capable of combining light. A dual functioning spacecraft can therefore both collect and combine light. However, to simplify the model it is assumed that both tasks can not be completed at the same time. To form the minimum of one baseline, each system is required to have at least two functioning spacecraft capable of collecting light and one additional functioning spacecraft capable of combining light to be considered in a working state. The requirement that a dual functioning spacecraft cannot both collect and combine light at the same time implies that each system must have a total of at least three functioning spacecraft (two collecting light plus one combining light) to be operable.

Each individual spacecraft is modeled in two parts: optics and bus. The combining spacecraft is modeled as combining optics plus a bus, the collecting spacecraft is modeled as collecting optics plus a bus, and the dual functioning spacecraft is modeled as combining optics, collecting optics, and a bus. Any of these components may fail throughout time. Failure rates are given by the user as an input for each set of optics (combining and collecting) and each type of bus (combining, collecting, and dual functioning). The combining and collecting spacecraft are considered in a failed state if either the optics or bus fails. The dual functioning spacecraft is switched to a combining spacecraft if it's collecting

optics fail, and is switched to a collecting spacecraft if it's combining optics fail. The dual functioning spacecraft can also transfer directly to a failed state if the bus fails.

With this simple model of a separated spacecraft interferometry system, the productivity, cost, and reliability of each architecture can be estimated. The models used to make these estimates are discussed in further detail in Chapter 3.

2.3 Chapter Summary

This chapter has introduced the reader to the basic physical concepts behind interferometry systems. These concepts then led to the development of a simple model of an interferometer system which will be used throughout this research. The concepts and model developed in this chapter are needed to estimate the life cycle metrics - reliability, productivity, and cost - of each architecture being evaluated. The models used to create these estimates will be discussed in Chapter 3.

Chapter 3

MODEL DEVELOPMENT

A series of Matlab functions have been developed to automatically estimate the productivity, cost, and reliability of a separated spacecraft interferometer system. The productivity of a system is in this case defined as the expected total number of images produced by the end of the mission lifetime. The cost is defined as the total life cycle cost of the system, including manufacturing of spacecraft and operations. The reliability is defined as the probability that the system is in a working state at the end of the mission lifetime. Each of the models used to estimate these three metrics depend on the state-transition matrix. Therefore, a method of automatically generating this matrix for any architecture is needed in order to automatically estimate any of these three metrics for the given architecture.

This chapter will first discuss the method used to automatically generate this state-transition matrix. Next, the model used to estimate each of these life cycle metrics - productivity, cost, and reliability - will be discussed in detail. Finally, the results of using all three of these models will be shown for an example case study.

3.1 State-transition Matrix

The state-transition matrix, also called the *A* matrix, defines both the states of a system and the rate at which the system will transition from one state to the next. A state of the system is defined by the number of operational spacecraft of each type (dual functioning, combining, and collecting). Therefore, the state of the system changes as failures occur in

spacecraft. The A matrix is found by analyzing the Markov model of a system. If P is defined as the vector of probabilities of being in each state of the system at any time, the A matrix is defined as:

$$\frac{dP(t)}{dt} = AP(t) \tag{3.1}$$

The state-transition matrix is essential in calculating the probability of being in each state of the system, and therefore is also needed to calculate several important parameters of the entire system, such as productivity, cost and reliability.

3.1.1 Automatic Generation of State-transition Matrix

Prior to this work, if any given architecture was to be analyzed using the state-transition matrix, this matrix needed to be calculated by hand and then entered into a Matlab file. When the state-transition matrix was later needed in analysis routines, the hand-entered and hand-calculated matrix was simply looked up from this Matlab file. In addition, the number of spacecraft acting as collectors for any given state needed to be entered by hand. Hand calculation of A matrices can introduce human errors, such as forgetting an entry or a negative sign. It is also possible to introduce errors when the state-transition matrices are manually copied into the Matlab file. In addition to errors, calculating the A matrix by hand takes a lot of time, especially for complicated systems (i.e. creating an A matrix for a system with 30 spacecraft that could work down to 3 spacecraft would be practically impossible due to the size of the matrix). Using this manual process, if it was desired to analyze any given architecture, that particular architecture's state-transition matrix would need to have been previously calculated. If the A matrix was not previously entered, it would need to be entered before the productivity, reliability, or cost could be calculated. As stated earlier, calculating and entering the state-transition matrix requires a lot of time. If the A matrix could be calculated and entered automatically, it would be possible to reduce potential for errors, save time, and have the ability to analyze any architecture. This leads to the ability to provide a much more thorough search of the design space, using

tools such as genetic algorithms or simulated annealing, instead of only searching through the designs that engineers have previously considered.

After calculating a large number of *A* matrices by hand, a pattern was noticed in the general *A* matrix. An *A* matrix can be created by looking at each state individually. Each row and column of the matrix corresponds to a different state. Figure 3.1 shows a very simple example of a state diagram, or Markov model, and the corresponding A matrix. Here d, m, and l are the failure rates of a dual functioning, combining, and collecting spacecraft respectively. The diagonal entries correspond to the ways in which the system could leave that state. In the example shown in Figure 3.1, three independent components could fail in state two. As a result, the diagonal entry for the second row would be minus one times the sum of the failure rates of each of the three components.

$$A(2, 2) = -(2d + m + l) \tag{3.2}$$

If a component fails in a given state and the system is still operating but in a different state, then the column entry of the new state's row would contain the rate at which this process occurs. Consider the example shown in Figure 3.1, where when a component fails the system transitions from the second to the third, fourth, or fifth state. In the A matrix representation, the third, fourth, or fifth row and second column entry would be the failure rate of the failed component. For example, when a collecting spacecraft fails, the system transitions from the second to the fifth state. Therefore, the fifth row and second column entry of the A matrix is the failure rate of the collecting spacecraft, or l.

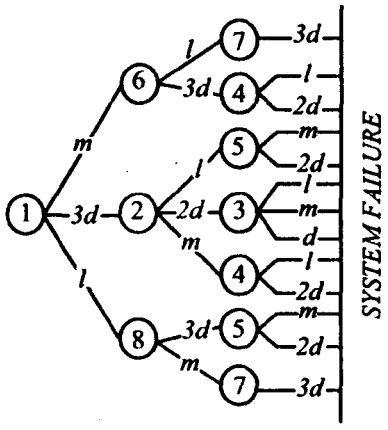
$$A(5, 2) = l \tag{3.3}$$

$$A(4, 2) = m \tag{3.4}$$

Since there are two identical components in the system, and if either one of them fails the system is considered in the third state, the corresponding third row and second column entry of the A matrix would be two times the failure rate of the component.

$$A(3, 2) = 2d \tag{3.5}$$

Any entry of the matrix that is not on the diagonal and does not connect one state to another in this regard is simply zero.



A=

-3*d-m-l	0	0	0	0	0	0	0
3*d	-2*d-m-l	0	0	0	0	0	0
0	2*d	-d-m-l	0	0	0	0	0
0	m	0	-2*d-l	0	3*d	0	0
0	l	0	0	-2*d-m	0	0	3*d
m	0	0	0	0	-3*d-l	0	0
0	0	0	0	0	l	-3*d	m
l	0	0	0	0	0	0	-3*d-m

Figure 3.1 Markov Model and corresponding *A* matrix for a sample system of three dual functioning spacecraft, one combining spacecraft, and one collecting spacecraft

Once the pattern has been identified, the main challenges in automating the process of creating the *A* matrix are defining the states, checking if a new state has been previously defined or not, knowing when a failure causes the system to move to a new state, and identifying when the entire system has failed. Additionally, a tree-type of structure needs to be implemented to be certain that every state has been analyzed.

The main variables in the automatic *A* matrix generation recursive Matlab function, *"state.m"*, are *a*, *dml*, *dual*, *com*, *col*, *last_state*, *f_d*, *f_m*, and *f_l*. The variable *a* is the state-transition matrix itself and is both an input and an output to the function. The variable *dml* is a matrix containing the state information, and is also both an input and an output of the function. As in the *A* matrix itself, each row of the *dml* matrix corresponds to a particular state. In the *dml* matrix, the columns correspond to the number of dual functioning spacecraft, combining spacecraft, and collecting spacecraft in that state respectively. Therefore by checking if the one-by-three vector consisting of the number of dual functioning spacecraft (*d*), the number of combining spacecraft (*m*), and the number of collecting spacecraft (*l*) in a particular state is already a row of the *dml* matrix, it is possible to see if that given state has previously been defined. If this vector is not already a row of the dml matrix, the state should be added as a new state. The variable *last_state* is an input to the function, and is the number of the previous state, from which the current state was derived. This allows the entries for all states to be entered in both the correct rows and columns. The variables *dual*, *com*, and *col* are the number of dual functioning spacecraft, combining spacecraft, and collecting spacecraft respectively in the current state, and the variables *f_d*, *f_m*, and *f_l* are their respective failure rates.

The function *"state.m"* is recursive, meaning that it calls itself within the function. The recursive process produces the tree structure mentioned above. The function has built in rules that decide whether a given number of each type of spacecraft is acceptable and could be a new state, or if the system has failed. Consider a system that requires two collecting spacecraft and one combining spacecraft to be functional. The operational rules for such a system are:

1. *The number of spacecraft acting as collecting spacecraft must be greater than or equal to two. This includes both the collecting spacecraft and the dual functioning spacecraft.*
2. *The number of spacecraft acting as combining spacecraft must be greater than or equal to one. This includes both the combining spacecraft and the dual functioning spacecraft.*

3. *Since the dual functioning spacecraft cannot collect and combine light at the same time, and since both two collecting spacecraft and one combining spacecraft must be working for the system to be operational, the total number of spacecraft must be greater than or equal to three.*

If from any given state the system can lose a dual functioning spacecraft and still meet all the criteria for an operable system, and this is the first time the state has been defined, then *"state.m"* is called with the same number of combining and collecting spacecraft and one less dual functioning spacecraft, and with *last_state* set at the number of the current state, *state_num*. The additional requirement of only calling *"state.m"* if this is the first time a state has been defined allows the algorithm to follow each branch of the tree to system failure only once. If this rule is not included, the system will repeat analyses through the same branches many times. While it is important to make sure that every possible way of reaching a given state is explored and analyzed, once a particular state has been reached, the path from that state to system failure will always be the same, no matter how the state was originally reached. Therefore, this rule was included to avoid excess analysis. Tests were done on many sample systems, both with this final rule and without. While the state-transition matrix for both cases were identical, the time required to generate the state-transition matrix was dramatically reduced in the cases in which the rule of only calling *"state.m"* if the current state was not previously defined was used. This can be seen in Table 3.1. This process is shown below, implemented in the *"state.m"* source code. Note that in the *"state.m"* source code *d, m,* and *l* are the number of dual functioning, combining, and collecting spacecraft respectively.

```
d_new = d-1;

if d+l > 2 & d+m > 1 & d+m+l> 3 & d >= 1 & state_num >...
number_of_states

[a,dml]=...
state(a,d_new,m,l,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end
```

TABLE 3.1 Comparison of *A* matrix and time needed to analyze *A* matrix when only calling *"state.m"* if current state was not previously defined (with extra rule) and always calling *"state.m"* if operational rules hold (no extra rule).

Duals	Combiners	Collectors	Size of *A* matrix (nxn)	Number of Entries in *A* matrix	Number of calls to *"state.m"* (no extra rule)	Number of calls to *"state.m"* (with extra rule)	Time to complete analysis (secs)) (no extra rule)	Time to complete analysis (secs) (with extra rule)
1	1	1	1	1	0	0	1	1.081
2	2	2	27	95	285	68	7	5.898
3	3	3	89	387	40221	298	156	20.529
3	3	4	110	489	122242	379	537	25.687
3	4	3	108	478	102564	370	445	25.066
4	3	3	139	641	627776	502	3443	34.449
2	1	1	10	24	15	14	3	2.684
3	1	1	25	86	219	61	6	5.468
4	1	1	47	188	3007	141	17	10.795
5	1	1	77	333	41388	256	145	17.646
6	1	1	116	527	576285	411	2652	27.029
2	2	1	16	47	57	31	4	3.725
2	1	2	18	57	85	39	4	4.086
3	2	1	36	135	812	99	9	7.901
3	1	2	38	145	1058	107	10	8.382
3	1	3	51	205	3630	154	19	11.427
5	2	1	101	452	165693	351	673	23.484
5	3	1	125	571	532871	446	2618	30.624
7	1	1	165	776	8134937	611	55639	41.56

This procedure is repeated for cases involving the loss of a combining or collecting spacecraft. The rules of operation can easily be changed to account for modified, or completely different, systems. As an example, during development the dual functioning spacecraft model was changed such that instead of simply a dual functioning spacecraft failing, instead either the combining optics, collecting optics, or bus would fail. In this new model, if the combining optics failed, the dual functioning spacecraft becomes a collector, if the collecting optics fail the dual functioning spacecraft becomes a combiner, and if the bus fails the spacecraft is lost. Prior to the automatic state-transition matrix tool, this change in the system model would have required entirely new *A* matrices to be developed

and entered by hand for every system under consideration. However, with this automatic tool, no new state-transition matrices were required to be analyzed by designers, since the tool automatically generates this matrix for any system when the tool is run.

Since the *a* and *dml* matrices are both inputs and outputs in each call to *"state.m"*, they are continuously updated. If losing any one of the components causes the system to fail the tests for operational ability, the current state must lead directly to system failure if that component fails, and *"state.m"* is not called again.

A few initial conditions must be entered before using *"state.m"* to automatically generate the *A* matrix. First, given an architecture, the first state's entry must initialize the *A* matrix by entering in the first row and column entry. The variable *last_state* must be initialized to the first state, or one. The *dml* matrix is initialized by setting the first row equal to all zeros. This enables *"state.m"* to identify if this is the first call to the function, or if it is one of the recursive calls. This ensures that the first state can also be used to call *"state.m"*, and therefore that all states have been analyzed. However the call to *"state.m"* for the first state only needs to check if components can fail and recall *"state.m"*, but does not need to fill in the *A* matrix, since it has already been initialized with the first state. Therefore it is important to be able to identify the first call to *"state.m"*.

Once the initial conditions are entered, one call to *"state.m"* will automatically produce the full *A* matrix and the full *dml* matrix. The *dml* matrix can then be used to calculate the number of spacecraft in each state acting as collecting spacecraft, and generate the number of baselines for each state. With *"state.m"* no *A* matrices will need to be generated or entered by hand, and any architecture can be analyzed immediately. Please see Appendix A for the source code for *"state.m"*.

3.1.2 Verification

Several automatically generated *A* matrices were checked against manually generated matrices, and shown to be identical. The productivity results, in terms of the expected

total number of images a system will produce by the end of the mission lifetime, from the automatic state-transition matrix generation were then compared to productivity results using the previously entered manual state-transition matrices, to check for errors in the program. Please see Section 3.2 for a discussion of the productivity model used for this test. All acceptable combinations of the three types of spacecraft, with a total of six spacecraft, were analyzed. The original comparison can be seen in the blue and maroon colored bars in Figure 3.2. The comparison produced identical results in most cases. A few specific cases, however, produced different productivity results for the different methods. A closer look at the *A* matrices of the four cases that differed resulted in the discovery of errors in the manually entered *A* matrices of all four cases. These errors included forgetting a negative sign, missing entries, and entries in the wrong places. The errors produced significantly different results from the corrected state-transition matrix. Once these state-transition matrices were corrected, the two methods were compared again, and proved to be identical, as can be seen in the blue and yellow bars in Figure 3.2. The results show that the automatic *A* matrix generating code provided the correct *A* matrix for all 17 cases checked, and also provided more accurate and reliable results than the manually generated cases.

3.2 Productivity Model

To find the expected total number of images a SSI system will produce, it is necessary to first find the productivity rate, or number of images per unit time, that each state of the system is capable of. This productivity rate can then be integrated through time, taking into account the probability of being in each state, to find the expected total number of images. The productivity rate for a given state is a function of the number of operational collecting spacecraft in that state, and therefore the number of independent baselines in the system. There is one independent baseline per every pair of collecting spacecraft that has a combining spacecraft located equal distance from each collecting spacecraft in the pair. The number of independent baselines, *Nb*, in a given state is calculated using Equation 3.7, where n is the number of spacecraft capable of collecting light in the given state. The

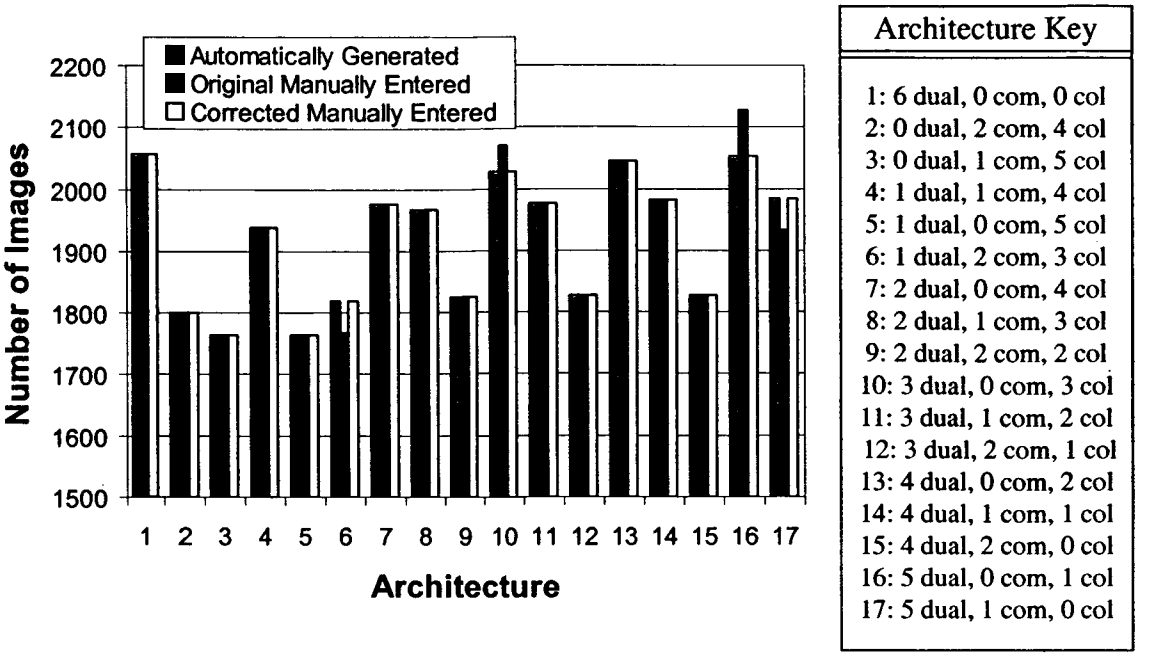


Figure 3.2 Comparison of productivity calculated using the automatically generated *A* matrix, the original hand-entered *A* matrix, and the corrected hand-entered *A* matrix

number of spacecraft capable of collecting light in a given state is defined as the number of collecting spacecraft plus the number of dual functioning spacecraft, minus one if there are no combining spacecraft, as seen in Equation 3.6. The subtraction of one if there are no combining spacecraft accounts for the fact that one dual functioning spacecraft then needs to combine light, and can therefore by definition not collect light. Please see Section 2.2 for a discussion of the dual functioning spacecraft model used here. Each baseline can provide one pair of UV points, or a fringe pattern. See Section 2.1 for a discussion on UV points and fringe patterns. Assuming the number of pairs of UV points needed to collect an image is known (and constant per image), the number of configurations needed in each state to collect one image is the number of pairs of UV points needed divided by the number of baselines per configuration, as shown in Equation 3.8. The time needed to take an image in each state can then be calculated as the number of pairs of UV points needed multiplied by the time to take one pair plus the number of configurations needed multiplied by the time needed to switch configurations plus some overhead time, as seen in Equation 3.9. The imaging, or productivity, rate of the given state is then the inverse of this time to take one image, as shown in Equation 3.10.

$$n_i = \left.\begin{matrix} dual_i + col_i \\ (dual_i - 1) + col_i \end{matrix}\right\} \begin{matrix} (com_i \geq 1) \\ com_i < 1 \end{matrix} \tag{3.6}$$

$$Nb_i = \frac{n_i(n_i - 1)}{2} \tag{3.7}$$

$$N_{c(i)} = ceil(N/(Nb_i)) \tag{3.8}$$

$$T_i = Nc_o + N_{c(i)}c_o + O_t \tag{3.9}$$

$$C_i = \frac{1}{T_i} \tag{3.10}$$

In Equations 3.6 through 3.10, $dual_i$ is the number of dual functioning spacecraft, col_i is the number of collecting spacecraft, and com_i is the number of combining spacecraft for the i^{th} state, n_i is the total number of collecting spacecraft in the i^{th} state, $N_{c(i)}$ is the number of configurations needed in that state, N is the number of pairs of UV points needed, Nb_i is the number of baselines in the i^{th} state, c_o is the time to both take one independent difference (pair of UV points) and to move a configuration, O_t is the overhead time, T_i is the time to take an image in the i^{th} state, and C_i is the imaging rate of the i^{th} state. N, c_o, and O_t are all constant parameters in this problem and are not changing from one state to the next.

Two different methods have been developed to calculate the expected total number of images a system will produce in a lifetime, in the event of failures, once the imaging rate for each state of the system has been found. The first method discussed uses a discrete version of the state-transition matrix and a defined time-step to sum up the expected number of images throughout the mission duration. The second method uses Laplace transforms to integrate through the mission lifetime and find the total expected number of images directly from the state-transition matrix.

3.2.1 Discrete *A* matrix

In the first method of calculating the expected total number of images a system will produce in a lifetime, the probability of being in any given state at a given time is calculated using a numerical integration solution to Markov models [Babcock, 1986]. The method involves transforming the A matrix from a continuous time matrix to a discrete time matrix, M, as shown in Equation 3.11, where Δt is the duration of the time-step and I is the identity matrix.

$$M = I + A\Delta t \tag{3.11}$$

The vector of the probabilities of being in each state at a given time, $P(t)$, can then be calculated as this M matrix multiplied by the probability of being in that state from one time step before, as shown in Equation 3.12.

$$P(t) = MP(t-1) \tag{3.12}$$

Looking at each time-step individually, the imaging rate for that time-step can be calculated by summing the imaging rate for each state multiplied by the probability of being in that state for the given time period, as shown in Equation 3.13. The number of images taken in that time step is then simply the imaging rate for that time step times the duration of the time step. The number of images can then be summed over all the time steps to get the expected total number of images for the mission, as shown in Equation 3.14.

$$c_t = \sum_i C_i P_i(t) \tag{3.13}$$

$$NoI = \sum_t c_t \Delta t \tag{3.14}$$

In Equations 3.13 and 3.14, c_t is the imaging rate of the entire system at time t, Δt is the time step, C_i is the imaging rate of state i, $P_i(t)$ is the probability that the system is in state

i at time *t*, and *NoI* is the expected number of images produced by the system over the entire lifetime.

3.2.2 Laplace Methods

The second method to find the total number of expected images a system will produce throughout its lifetime, taking into account failures, is to use Laplace methods to integrate the imaging rate directly. Three general Laplace transform rules are used in the following derivation and are shown in Table 3.2 [Strang, 1986].

TABLE 3.2 General Laplace rules

Time Domain	Laplace Domain
$e^{at}f(t)$	F(s-a)
f'(t)	sF(s)-f(o)
1	1/s

The derivation for the total number of expected images a system will produce begins with the definition of the state-transition matrix, Equation 3.1. This definition is transformed to the Laplace domain, manipulated, and transformed back to the time domain to obtain a single equation for the probability vector at a given time, *t*, in terms of the state-transition, or *A* matrix, and the initial probability vector. The probability vector is a column vector with *n* rows, when *n* is defined as the number of possible operational states that exist for the given system. Each row in this vector corresponds to the probability that the system is in that particular state. The initial probability vector is a vector of zeros, with a one in the first entry. This is due to the fact that the system is assumed to be working and in its initial state at time *t=0*, with a probability of 1. The productivity for any given time, *t*, can be found by multiplying the imaging rate, C_i, found in Equation 3.10, for each state by the probability that the system is in that given state. If the imaging rates are arranged in a productivity vector, *C*, defined as a row vector of the imaging rate in each state, then the productivity for any given time is given by the productivity vector times the probability

vector, and is a scalar. This productivity can then be integrated through the lifetime of the system to find a total expected productivity [Belanger, 1995; Selby, 1971]. This derivation is shown in Equations 3.15 through 3.22.

$$\frac{d}{dt}P(t) = AP(t) \tag{3.15}$$

$$sP(s) - P_o = AP(s) \tag{3.16}$$

$$(sI - A)P(s) = P_o \tag{3.17}$$

$$P(s) = (sI - A)^{-1}P_o \tag{3.18}$$

$$P(t) = e^{At}P_o \tag{3.19}$$

$$z(t) = CP(t) = Ce^{At}P_o \tag{3.20}$$

$$z_{tot} = \int_0^{life} z(t)dt = \int_0^{life} Ce^{At}P_o dt = C\frac{1}{A}e^{At}P_o\Big|_0^{life} \tag{3.21}$$

$$\therefore z_{tot} = CA^{-1}(e^{A \cdot life} - I)P_o \tag{3.22}$$

In Equations 3.15 through 3.22, *life* is the mission lifetime, P is the probability vector, P_o is the initial probability vector, C is the productivity vector, A is the state-transition matrix, $z(t)$ is the productivity at time t, and z_{tot} is the total expected number of images.

3.2.3 Comparison of Methods

The two methods described above for calculating the total number of expected images a system will produce in the event of failures were tested on various architectures and compared against one another to check for accuracy in both methods. Seven different architectures were tested with both methods and the results were compared. The largest

difference between the two methods was only 0.056% of the results found from the Laplace methods. The average difference between the two methods was only 0.6 images, or 0.048%. These results are shown in Table 3.3 below. The similarity between the two methods' results leads to a high confidence in the accuracy of both methods.

TABLE 3.3 Comparison of discrete *A* matrix and Laplace methods to find the productivity of a system

Combiners	Collectors	Dual Functioning	NoI: Discrete A matrix	NoI: Laplace	Delta NoI	%Delta NoI
2	2	2	1621.4	1620.5	0.9	0.056
1	1	1	564.3	564.1	0.2	0.035
1	2	1	1143	1142.4	0.6	0.053
1	1	2	1141.3	1140.8	0.5	0.044
2	1	1	707.9	707.6	0.3	0.042
1	1	3	1553.9	1553.1	0.8	0.052
3	3	3	1980.1	1979	1.1	0.056
Average			**1244.6**	**1243.9**	**0.6**	**0.048**

Once both methods were checked against one another, it was decided to use the Laplace method in future analyses. This decision was based on the number of equations and "if-loops" required in the coding of both methods. Due to the large number of "if-loops" in the discrete *A* matrix method, this method takes more time and computational effort to compute than the Laplace method, and was therefore not used in further, more computationally intensive analyses.

3.2.4 Benchmarking

While no data is available to truly benchmark this simulation code, the results returned by the code do logically make sense, as shown by the following parameter study. The design vector for this problem includes the number of each type of spacecraft (combining, collecting, and dual functioning) and the money spent to improve the reliability of each component (combining optics, collecting optics, and bus). Please see Section 3.4.2 for a discussion of how the money spent to improve the reliability of components affects the

failure rates of the spacecraft, and therefore the state-transition matrix. An initial architecture of two of each type of spacecraft with no money spent on improving any component's reliability was used to calculate the initial total number of expected images. From here, different aspects of the architecture were changed, and the calculated expected total number of images for these new architectures were compared to the initial expected total number of images. The results can be seen in Table 3.4 below.

TABLE 3.4 Parameter study of results returned from productivity model

Number of dual func. spc.	Number of comb. spc.	Number of collec. spc.	Money to improve combining optics rel. ($M)	Money to improve collecting optics rel. ($M)	Money to improve bus reliability ($M)	Expected number of images
2	2	2	0	0	0	1621
3	2	2	0	0	0	1841
2	3	2	0	0	0	1644
2	2	3	0	0	0	1835
2	2	2	100	0	0	1647
2	2	2	0	100	0	1794
2	2	2	0	0	100	1821

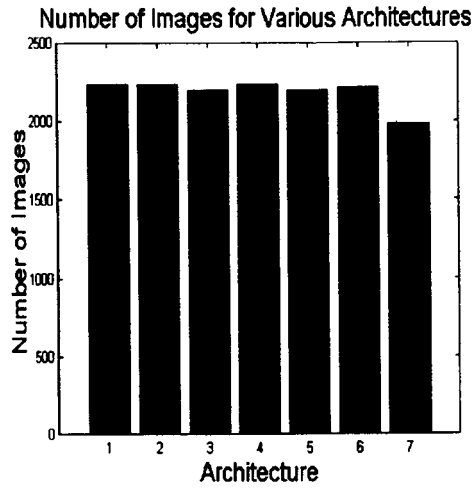
While an increase in the number of any spacecraft type does provide an increase in the expected number of images, the dual functioning spacecraft have the largest effect. This makes sense since these spacecraft can be used as either combiners to keep the system functioning or collectors to improve the productivity of the system. The combining spacecraft had the least effect on the number of images. This also makes sense since only one combining spacecraft is used at a time, and in order for three combiners to make an impact on the system, two combiners would have already had to fail. Following this same logic, when an extra dual functioning spacecraft is added to the architecture it will act as a collecting spacecraft the vast majority of the time, and will only act as a combining spacecraft when the two original combining spacecraft have already failed. This explains the similarity in the expected total number of images for when a dual functioning spacecraft is added and when a collecting spacecraft is added. In addition, improving any of the com-

ponent reliabilities also increases the expected total number of images. In this case, improving the bus reliability has the largest impact. This is logical since the bus affects all three types of spacecraft, whereas both of the sets of optics only impact two types of spacecraft each. For the same reasoning as above, improving the collecting optics reliability has a much greater impact on the number of images than improving the combining optics reliability.

3.2.5 Case Studies

Two case studies were carried out to demonstrate the ability of the model generation functions to adapt to varied systems. Both case studies assumed no money was spent to improve the reliability of components. A case study of the productivity of different combinations of types of spacecraft, with fifteen total spacecraft, was carried out using the automatic model generating code. This case considered four collecting spacecraft and one combining spacecraft to be the minimum required for system operations. With hand-entered A matrices, this case study could have taken days, or even weeks to complete, since an A matrix would have to be hand calculated for each combination of spacecraft. With such a large system, this A matrix is quite complex and large. Creating a Markov model of this system, and then creating an A matrix by hand from this model could take an enormous amount of time. In addition, even if one of the A matrices had been previously entered for the usual case of the system requiring only two collecting spacecraft to be functional, the A matrix would still have to be re-calculated since the rules of system operations had changed. These rules of system functionality are easily changed when using the automatic model generation by making the operational rules in the *"state.m"* source code reflect this change. Changing the rules to reflect system failure with fewer than four collecting spacecraft took only a few minutes, and this case study was run with less than half an hour of preparation time. The results of this case study can be seen in Figure 3.3.

In the next case study, the system can function with as few as two collecting spacecraft and one combining spacecraft. In this case study, however, the system requires one com-



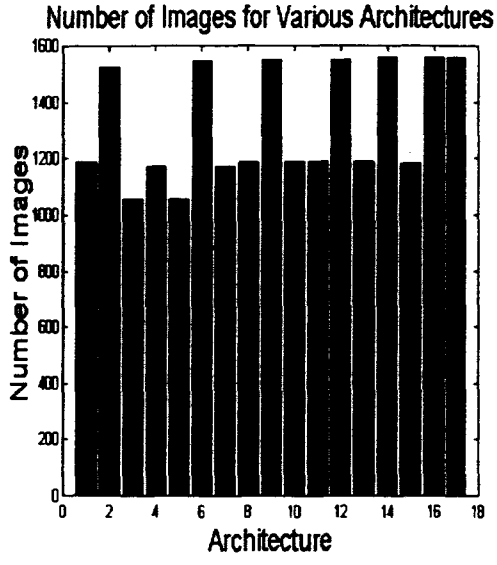
Architecture Key
1: 15 dual, 0 com, 0 col
2: 10 dual, 0 com, 5 col
3: 5 dual, 5 com, 5 col
4: 5 dual, 0 com, 10 col
5: 0 dual, 5 com, 10 col
6: 0 dual, 3 com, 12 col
7: 0 dual, 1 com, 14 col

Figure 3.3 Case study 1 - Different combinations of a total of 15 spacecraft. Systems considered operational down to 4 collecting spacecraft and 1 combining spacecraft.

bining spacecraft for every two collecting spacecraft. In other words, the spacecraft came in sets of two collecting spacecraft and one combining spacecraft. If there were only three collecting spacecraft in any given state, one of these spacecraft was simply a back-up and not used to produce images in that state. Each set of one combining spacecraft and two collecting spacecraft was considered a baseline. The number of images was then calculated in the same fashion as discussed previously. While this case study requires no new A matrices, it does require the method of calculating the number of baselines to be changed. This change also took very little time to complete and the case study was completed in approximately one hour. The case study was run on combinations of six total spacecraft, and the results can be seen in Figure 3.4. The results from this case study can also be compared to the values shown in Figure 3.2 to see the difference in productivity between the two definitions of baselines.

3.3 Cost Model

A cost model was developed to estimate the total life cycle cost of a separated spacecraft interferometer system. This model was developed for the purpose of comparing architectures that differ in only the number of each type of spacecraft and money spent to improve



Architecture Key
1: 6 dual, 0 com, 0 col
2: 0 dual, 2 com, 4 col
3: 0 dual, 1 com, 5 col
4: 1 dual, 1 com, 4 col
5: 1 dual, 0 com, 5 col
6: 1 dual, 2 com, 3 col
7: 2 dual, 0 com, 4 col
8: 2 dual, 1 com, 3 col
9: 2 dual, 2 com, 2 col
10: 3 dual, 0 com, 3 col
11: 3 dual, 1 com, 2 col
12: 3 dual, 2 com, 1 col
13: 4 dual, 0 com, 2 col
14: 4 dual, 1 com, 1 col
15: 4 dual, 2 com, 0 col
16: 5 dual, 0 com, 1 col
17: 5 dual, 1 com, 0 col

Figure 3.4 Case study 2 - Combinations of six total spacecraft. Number of baselines considered number of two collecting spacecraft, one combining spacecraft pairs.

the component reliabilities. Therefore factors that would have a large effect on the cost of a system, but which were constant among all architectures, such as mirror diameter size, were not considered. Only two factors were considered in estimating the relative cost of an architecture: manufacturing and operations cost.

When modeling the manufacturing cost, the fact that all architectures only varied by the number of each type of spacecraft was taken advantage of. Using this fact, all theoretical first unit costs of the components of each spacecraft (optics and bus) were assumed to be known. These theoretical first unit costs can be set by the user as inputs. The default values for the theoretical first units costs are $25M for combining optics, $15M for collecting optics, $20M for combining and collecting spacecraft buses, and $30M for dual functioning spacecraft buses. These values were chosen for their relationships with one another. The combining optics in any interferometer system are much more complicated than the collecting optics, and were therefore priced higher. The combining bus and collecting bus were priced equal since they are generic spacecraft buses. However, the dual functioning bus was costed higher than the other two buses to account for the added complexity needed when a spacecraft can perform either function. Lastly, the total theoretical first

unit cost is \$45M for a combiner, \$35M for a collector, and \$70M for a dual functioning spacecraft. This means that the cost of buying separate combining and collecting spacecraft (\$80M) is more than buying one dual functioning spacecraft (\$70M), but is still in the same order of magnitude. With the theoretical first unit cost of all components known, the theoretical first unit cost of each type of spacecraft can then be found. Once the theoretical first unit cost of each type of spacecraft is known, the total manufacturing cost for each type of spacecraft can be found using the total number of spacecraft of that type in the system and a learning curve savings. The learning curve slope can be input by the user, but is set as a default at 95%, as recommended in Wertz and Larson [Wertz and Larson, 1999]. The components are not affected by a learning curve savings individually, but only as total spacecraft savings. The manufacturing costs of each type of spacecraft are found in Equations 3.23 through 3.25.

$$DualCost = (DB_{TFU} + MO_{TFU} + LO_{TFU}) \times \left(dual^{\frac{\left(1-\left(\ln\frac{100}{m}\right)\right)}{\ln 2}} \right) \tag{3.23}$$

$$ComCost = (MB_{TFU} + MO_{TFU}) \times \left(com^{\frac{\left(1-\left(\ln\frac{100}{m}\right)\right)}{\ln 2}} \right) \tag{3.24}$$

$$ColCost = (LB_{TFU} + LO_{TFU}) \times \left(col^{\frac{\left(1-\left(\ln\frac{100}{m}\right)\right)}{\ln 2}} \right) \tag{3.25}$$

In Equations 3.23 through 3.25, DB_{TFU}, MB_{TFU}, LB_{TFU}, MO_{TFU}, and LO_{TFU} are the theoretical first unit costs in \$M of the dual-functioning spacecraft bus, combining bus, collecting bus, combining optics, and collecting optics respectively. *Dual, Com,* and *Col* are

the number of dual functioning, combining, and collecting spacecraft in the system respectively, and m is the learning curve slope percentage.

Operations costs are assumed to scale with the number of baselines. This is due to two factors: 1) size of clusters and 2) time between cluster reorientations. More baselines lead to larger clusters needed to keep in formation, which requires more effort than smaller clusters. Also, the larger the number of baselines, the faster the light can be collected, and therefore the sooner the cluster needs to be reoriented to begin a new image. While smaller clusters will need to switch configurations to gather more baselines and UV points, these configuration moves will be smaller and less complicated than the moves required to begin a new image. Since clusters with more baselines can collect more images in the same time, the operations cost for these clusters will be higher because there is more complicated cluster movement. A typical operations cost is given by the user as an input, in terms of cost per baseline per month. Once the state definitions are found from the productivity model, this typical cost can be turned into a vector of costs per month for each state of the system based on the number of baselines in that state. This vector can then be integrated, using the same procedure as described in Section 3.2.2, to find the total operations cost of the system.

$$CostPerState = ops \times Nb \tag{3.26}$$

$$OpsCost = CostPerState \cdot A^{-1}(e^{A \cdot life} - I)P_o \tag{3.27}$$

In Equation 3.26, *ops* is the baseline operations cost ($M/baseline/month) and *Nb* is a vector of the number of baselines in each state of the system. In Equation 3.27, A is the state-transition matrix, *life* is the total mission design lifetime (months), and P_o is the vector of probabilities of being in each state at the beginning of the mission.

Once the manufacturing and operations cost of a system are known, the total life cycle cost of the system can be calculated. This total cost is simply the manufacturing cost, plus

the operations cost, plus the amount of money spent to improve each component's reliability.

$$Cost = DualCost + ComCost + ColCost + OpsCost + Xmo + Xlo + Xb \quad (3.28)$$

In Equation 3.28, *Xmo, Xlo,* and *Xb* are the money spent to improve the reliability of the combining optics, collecting optics, and all three buses respectively ($M).

3.4 Reliability Model

There are two unique ways to improve the overall reliability of a complex system consisting of multiple components - increase redundancy of components or improve the initial reliability of these components. These two methods are both captured in the design vector of this problem. Redundancy in the system is captured by the number of each type of spacecraft (combining, collecting, and dual functioning). Improving reliability is captured by the money spent to improve the reliability of the components (combining optics, collecting optics, and bus). Therefore, a model estimating the effect of each of these methods needs to be developed. There are two steps to this model - estimating the reliability of a system given the number of each type of spacecraft and the failure rates of those spacecraft, and improving the failure rates, or reliabilities, of the components given a certain amount of money spent on improving the reliabilities.

3.4.1 Estimating Reliability

Once the state-transition matrix, or *A* matrix, is calculated, this matrix can be used to find the probability that the system is any state at any time (see Equations 3.15 - 3.19). The definition of reliability is the probability that the system is in a working state at the end of the mission lifetime. Since the probability that the system is in each working state at the end of the mission lifetime can be found from the state-transition matrix, the reliability is therefore the sum of these probabilities. In this way, the reliability of the system can easily be found, using very little information that was not already calculated for the productivity analysis. This process is shown in Equations 3.29 and 3.30.

$$P_{life} = e^{A \cdot life} P_o \tag{3.29}$$

$$Reliability = \sum_i P_{life}(i) \tag{3.30}$$

In Equations 3.29 and 3.30, P_{life} is a vector of the probabilities that the system is in each state at time $t=life$, *life* is the mission design lifetime of the system (months), P_o is the vector of probabilities that the system is in each state at time $t=0$, and A is the state-transition matrix.

3.4.2 Improving Reliability

It is possible to improve a component's reliability by either further testing or improving the design of the component. Each method takes additional money however. Therefore, a model is necessary to predict how much a component's reliability will improve if a given amount of money is spent on either of these activities.

The reliability of any component is always between zero and one. In addition, while a significant improvement in the reliability of a component may be improved when money is initially spent on this task, the closer the reliability of the component gets to one, the more difficult it is to improve this reliability further. The final reliability of any component, after money is spent to improve the reliability, should therefore asymptote to one. This relationship is captured in the model shown in Equation 3.31.

$$R_C = R_{OC} + (1 - R_{OC}) \left(1 - e^{-\left(\frac{X_C}{S}\right)} \right) \tag{3.31}$$

In Equation 3.31, R_C is the final reliability of the component, R_{OC} is the initial component reliability, X_C is the money spent to improve the component in millions of dollars, and S is a scale factor. Notice that the final component reliability will never be less than the initial component reliability, assuming that X_C is restricted to be greater than or equal to zero.

Additionally, if no money is spent to improve the reliability, X_C is zero and therefore the last term in Equation 3.31 is also zero, and the final reliability is equal to the initial reliabiltiy. As the money to improve the reliability is increased, the last factor of the second term in Equation 3.31 becomes closer to one, implying that the final reliability approaches one [Jilla, 2000]. This relationship can be seen graphically in Figure 3.5.

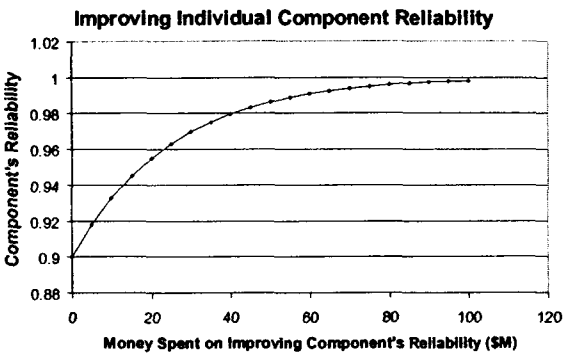


Figure 3.5 Model of how money spent to improve reliability is translated to actual reliability improvement

The model discussed here is implemented for three components of separated spacecraft interferometry systems - combining optics, collecting optics, and a generic bus. See Section 2.2 on page 44 for a discussion of the model used to describe these SSI systems and how these components fit together to form spacecraft. The initial reliabilities of each component and a general scale factor can be set by the user.

3.5 Results

Once the total expected number of images, life cycle cost, and reliability of a system are calculated, the performance of the system can be reported with three different outputs: the number of images, the cost per image, and the reliability of the system. Each output is important and captures a different aspect of the system. Eleven possible architectures, each with a total of four spacecraft and no money spent to improve the reliability of com-

ponents, were modeled using the previously described productivity, cost, and reliability models. The results are shown in Figure 3.6, with an architecture key shown in Table 3.5.

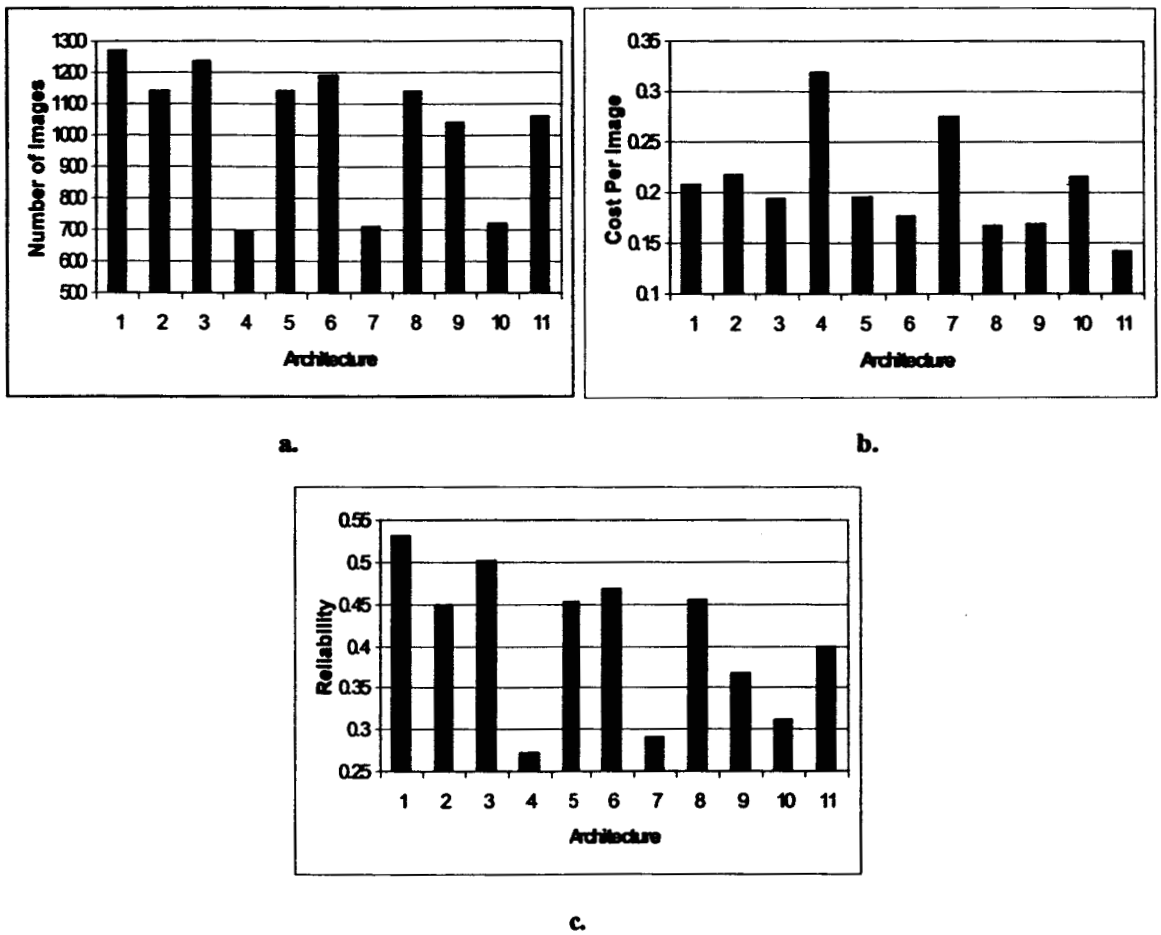


Figure 3.6 Productivity modeling results. a) Number of images. b) Cost per image. c) Reliability. See Table 3.5 for architecture key.

The results for this case study show that while architecture one (four dual functioning spacecraft, no combining or collecting spacecraft) produces both the highest number of images and highest reliability, architecture eleven (no dual functioning spacecraft, one combining spacecraft, and three collecting spacecraft) produces the lowest cost per image. This leads to the need to compare architectures based on a combination of number of images, cost per image, and reliability.

TABLE 3.5 Architecture key for case study shown in Figure 3.6

Architecture Key
1 - 4 dual, 0 com, 0 col.
2 - 3 dual, 1 com, 0 col.
3 - 3 dual, 0 com, 1 col.
4 - 2 dual, 2 com, 0 col.
5 - 2 dual, 1 com, 1 col.
6 - 2 dual, 0 com, 2 col.
7 - 1 dual, 2 com, 1 col.
8 - 1 dual, 1 com, 2 col.
9 - 1 dual, 0 com, 3 col.
10 - 0 dual, 2 com, 2 col.
11 - 0 dual, 1 com, 3 col.

3.6 Chapter Summary

This chapter has discussed the method used to automatically generate the state-transition matrix for any system. This matrix is necessary in order to estimate all three life cycle metrics. The method used to generate the state-transition matrix was tested and verified. In addition to allowing the analysis of any architecture, and not just the architectures a designer has previously thought of, this method also proved to reduce errors in the generation of these matrices. Next, the productivity model, in terms of the expected total number of images an architecture would produce by the end of the mission lifetime, was discussed in detail. This model was then bench-marked and tested in several varying case studies. The models and methods used to estimate the total life cycle cost of the system, including manufacturing and operations costs, were also discussed. Next, the method of estimating reliability, defined as the probability that the system is in a functioning state at the end of the mission lifetime, from the state-transition matrix was covered. In addition, a model to estimate the total increase in the reliability of a component for a given amount of money spent in an attempt to improve the initial reliability was developed. Finally, the results of using all three of these models on an example case study were given. It was shown that no one architecture, of the eleven analyzed, had the highest estimated productivity and reliability while still maintaining the lowest cost. This leads to the need to compare architec-

tures based on a combination of these metrics. The method for accomplishing this and the results will be discussed in Chapter 4.

Chapter 4

ARCHITECTURE COMPARISON BASED ON TOTAL PERFORMANCE

A system with high reliability has a higher probability of functioning throughout the mission lifetime. Therefore, systems with higher reliabilities will generally function longer than systems with lower reliabilities. In a similar manner, systems with high productivity will generally produce more images than systems with low productivity. Reliability or productivity alone does not make an acceptable system however. Neither a system that lasts for several years, but takes months to produce a single image, nor a system that can produce an image in just minutes, but only functions for a month will be funded or supported. In addition, a system which has the highest reliability and productivity possible, but would also cost double the allowable budget, will not be supported. Therefore, it is important to find a way to compare the total performance, by coupling reliability, productivity, and cost, of multiple systems in order to make informed decisions of which architecture is "best", or which family of systems should be examined in further detail.

This chapter will first discuss the method used to compare the total performance of different user defined architectures. Next, several case-studies showing the results of using this method will be presented. These case studies will first examine architectures defined only in terms of the number of each type of spacecraft, and will then move on to include money spent to improve component reliabilities in the design vector.

4.1 Total Performance

The major step needed to be able to compare architectures based on their total performance, including reliability, productivity, and cost, is to combine all three of these metrics into one total performance metric, or score. This was done initially only to compare different user defined architectures, and not to explore the entire design space such as what would be done with an optimization tool. For a discussion on the performance metric used in an optimization tool, where the entire design space is explored, please see Section 5.1.1 on page 105.

4.1.1 "Score" Metric Formulation

The performance, or "score" metric, is basically a weighted sum of the expected number of images, reliability, and cost per image for a given architecture. Please see Section 3.2 on page 55 for a discussion on the method of calculating the expected number of images, Section 3.3 on page 64 for a discussion on the method of calculating the cost per image, and Section 3.4 on page 68 for a discussion on the method of calculating the reliability for a given system. The expected number of images and the reliability are both "larger is better" metrics. That is to say that the more number of images a system produces and the higher the reliability of the system, the more advantageous the system is. Contrarily, the cost per image is a "smaller is better" metric since the less the cost of an image is, the more advantageous the system producing that image is. Since two of the three components were already "larger is better" metrics, it was decided to make the overall performance metric "larger is better" as well. Therefore, it was necessary to sum scaled and weighted versions of the expected number of images, the reliability, and the inverse of the cost per image.

It should be noted that the three components of the performance metric are of different orders of magnitude. The productivity, or expected number of images, is on the order of thousands. The reliability is always between zero and one. The cost per image is on the order of one tenth, and the inverse of the cost per image is on the order of ten. In addition,

when comparing two architectures it is more the relative difference, or percentage difference, between the values of these metrics that different architectures produce that matters, and not the absolute difference. For example, the difference between 2000 images and 2001 images is much less substantial of a difference than the difference between a reliability of 0.9 and 0.95. Therefore, each metric is scaled by the largest (or smallest in the case of cost per image) value of that metric calculated for any of the systems being compared. In this way, a system which produces the largest expected number of images will always have a value of one for the productivity portion of the performance metric. A system with the second largest expected number of images, but only by one or two images, will therefore have a value of near one, but not exactly one. This allows a distinction to be made between architectures, without ranking and forcing the second best performing system in one metric to be a given number of points below the first. Scaling each metric by the maximum value found also removes all units such that the relative "score" for each metric can be summed together. The maximum and minimum values of each metric can also be easily replaced with threshold values, if performance beyond those values is not any more beneficial.

In addition to scaling, the metrics are also weighted before being summed to produce the final performance metric. The total of the three weightings, one each on productivity, reliability, and cost, should sum to one. These weightings are user defined as inputs and can easily be adjusted to assess impact. The weightings are important to allow an individual program or designer to decide how important each aspect of the system is to their particular design. These weighting could vary widely from project to project but are necessary to capture the true needs of individual programs.

The preceding system of both scaling and weighting leads to the final "score" metric shown in Equation 4.1 for the i^{th} architecture being evaluated. It should be noted that the "score" for any architecture will always be between zero and one. If one architecture has the highest expected number of images, highest reliability, and lowest cost per image of all architectures evaluated, that architecture would receive a "score" of exactly one.

$$Score(i) = w_R \frac{Reliability(i)}{MaxReliability} + w_P \frac{Productivity(i)}{MaxProductivity} + w_C \frac{MinCostPerImage}{CostPerImage(i)} \quad (4.1)$$

In Equation 4.1, *Score(i)* is the total performance metric, *Reliability(i)* is the reliability, *Productivity(i)* is the number of images, and *CostPerImage(i)* is the cost per image, all for the i^{th} architecture evaluated. In addition, w_R, w_P, and w_C are the weighting values assigned to the reliability, productivity, and cost per image respectively, while *MaxReliability, MaxProductivity,* and *MinCostPerImage*, are the best values of each of the three metrics among all the architectures compared.

It is important to note that each of the three terms in Equation 4.1 captures a unique system property. The reliability is defined as the probability that the system is in a working state, with at least one spacecraft capable of combining light and two spacecraft capable of collecting light functional at the end of the mission lifetime. This metric is important if the system must function for a given amount of time in order to have the mission considered successful.

While the reliability captures the amount of time the system is expected to function, it contains no information on how productive the system is during that lifetime. The productivity is defined as the expected total number of images the system will produce by the end of its mission lifetime. Recall that the imaging rate of the SSI systems modeled here scales with the number of spacecraft capable of collecting light.

Systems with a large number of collecting spacecraft and only one spacecraft capable of combining light will be very productive while they are functioning; however, these systems will fail as soon as the combining spacecraft fails, causing the system to have a relatively low reliability. If two systems that have the same cost are analyzed, one of which contains redundant combining spacecraft and therefore fewer collecting spacecraft, and the other of which contains only one combining spacecraft and more collecting spacecraft, the first system will have a lower average imaging rate, but will function longer than the second system. Since productivity, as measured by the total expected number of images,

can be thought of as the average imaging rate multiplied by the time the system is functioning, these two systems may have nearly the same productivity. However, the first system will have a higher reliability. Conversely, two different systems, again of equal cost, can have similar reliabilities but varying productivities. Consider two systems, each with redundant combining spacecraft, but the first system with as many redundant collecting spacecraft as the budget allows and the second with fewer collecting spacecraft and the extra money in the budget spent on improving the reliability of these collecting spacecraft. These two systems may have very similar reliabilities, but the first system will have a higher average imaging rate, and will therefore produce more images than the second. Reliability and productivity are therefore unique metrics, since it is possible to have both two systems with similar productivities and different reliabilties, and two systems with similar reliaiblities and different productivities.

The third term in Equation 4.1, cost per image, is also a unique metric. The cost of a system depends on the number of each type of spacecraft and the money spent to improve the reliabilities of components. The more total spacecraft a system has, the more complex it is, and therefore more expensive it is to operate. In addition, dual functioning spacecraft cost more than combining spacecraft, which in turn cost more than collecting spacecraft. One of the many ways to improve reliability and productivity at the same time is to use dual functioning spacecraft. These spacecraft can collect light, making the system more productive while another spacecraft is combining light, and can switch over to combining light, allowing the system to continue to function if the spacecraft that was previously combining light fails. However, since the dual functioning spacecraft cost the most of all three types of spacecraft, this system will be more expensive than if the dual functioning spacecraft was replaced by either an additional combining or collecting spacecraft. This interaction between improved productivity and reliability and increased cost of the system can be captured by including one of two metrics in the total performance metric; cost or cost-effectiveness. In this case, cost-effectiveness is defined as the cost per image. While either of these metrics, cost or cost-effectiveness, could be used in the total performance metric, the research presented here used cost-effectiveness, since it captures the relative

increase in cost compared to an increase in the number of images. If desired, changing the final term in Equation 4.1 to cost instead of cost per image would be a trivial change and one that could be explored in the future.

4.1.2 Results

The reliability and "score" metric described above were analyzed for a case study involving 16 different architectures, each with a total of six spacecraft, with no money spent on improving any component reliabilities. The specific architectures can be seen in Table 4.1, and the results of the case study can be seen in Figure 4.1. Note that for this case study, w_R and w_P were set to 0.3 and w_C was set to 0.4.

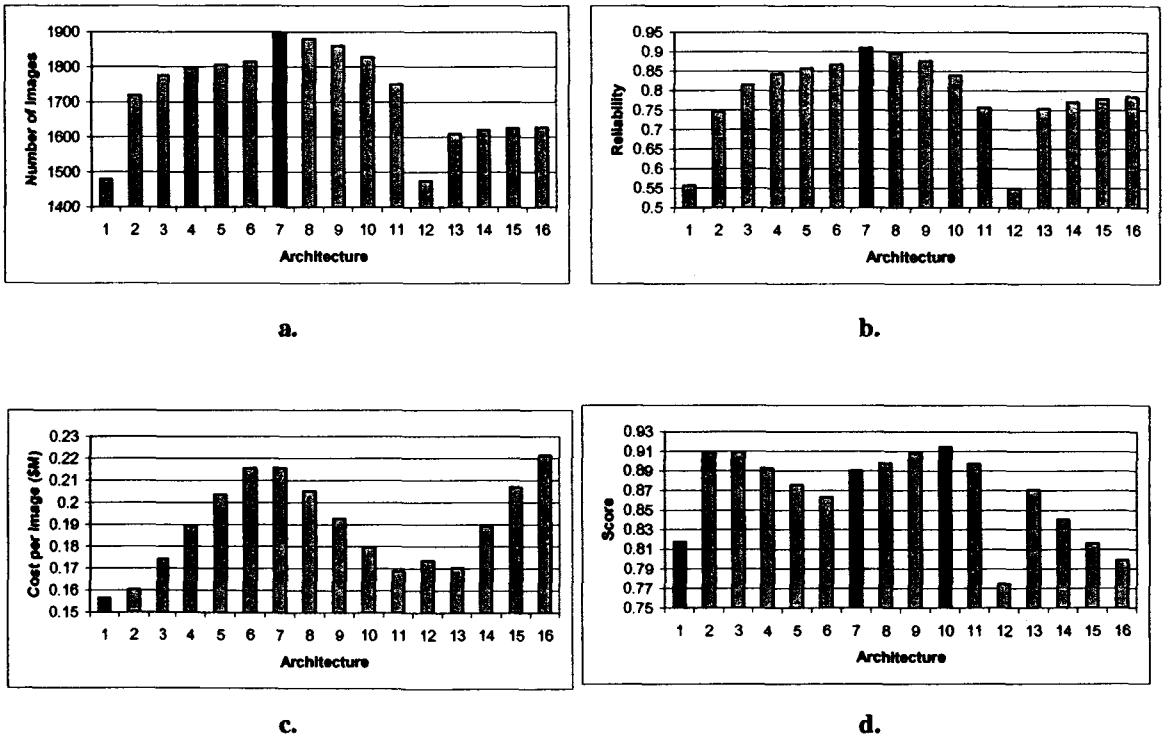


Figure 4.1 Case Study 1 - combinations of 6 total spacecraft. a) Number of Images b) Reliability c) Cost per Image d) "Score". See Table 4.1 for Architecture Key

TABLE 4.1 Architecture key for case study 1.

Architecture Key
1 - 0 dual, 1 com, 5 col.
2 - 1 dual, 1 com, 4 col.
3 - 2 dual, 1 com, 3 col.
4 - 3 dual, 1 com, 2 col.
5 - 4 dual, 1 com, 1 col.
6 - 5 dual, 1 com, 0 col.
7 - 6 dual, 0 com, 0 col.
8 - 5 dual, 0 com, 1 col.
9 - 4 dual, 0 com, 2 col.
10 - 3 dual, 0 com, 3 col.
11 - 2 dual, 0 com, 4 col.
12 - 1 dual, 0 com, 5 col.
13 - 1 dual, 2 com, 3 col.
14 - 2 dual, 2 com, 2 col.
15 - 3 dual, 2 com, 1 col.
16 - 4 dual, 2 com, 0 col.

In this case study, architecture seven (six dual functioning spacecraft, no combining or collecting spacecraft) has the highest expected total number of images and reliability. Architecture one (zero dual functioning spacecraft, one combining spacecraft, and five collecting spacecraft) has the lowest cost per image. However, architecture seven is very expensive, while architecture one is neither productive nor reliable. Architecture ten (three dual functioning spacecraft, no combining spacecraft, and three collecting spacecraft) has the best combination of all three metrics, even though it did not perform the best in any of the individual categories.

The same case study that was seen in Section 3.5 on page 70 was carried out again, this time with the total "score" calculations. The results can be seen in Figure 4.2. In this particular case study, the architecture with the lowest cost, architecture eleven (no dual functioning spacecraft, one combining spacecraft, and three collecting spacecraft), was also the architecture with the best total performance. It was not however the same as the architecture with both the highest productivity and highest reliability - architecture one (four dual functioning spacecraft, no combining or collecting spacecraft).

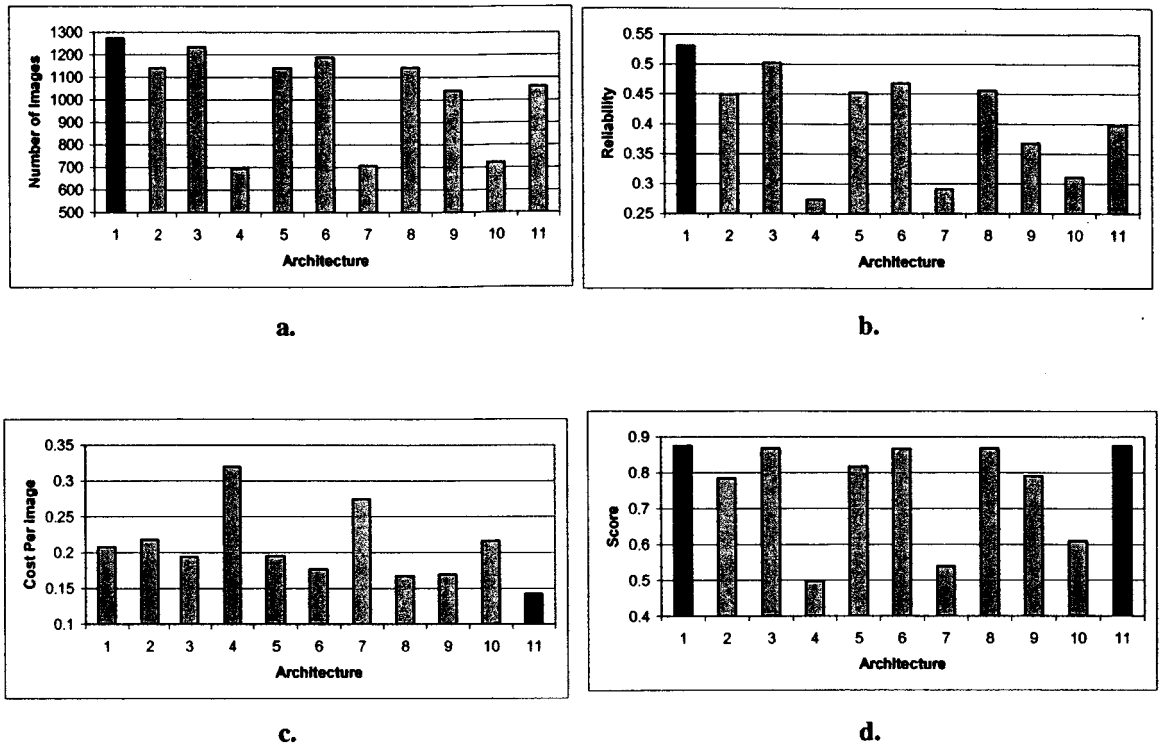


Figure 4.2 Case Study 2 - combinations of 4 total spacecraft. a) Number of Images b) Reliability c) Cost per Image d) "Score". See Table 4.2 for Architecture Key

TABLE 4.2 Architecture key for case study 2.

Architecture Key
1 - 4 dual, 0 com, 0 col.
2 - 3 dual, 1 com, 0 col.
3 - 3 dual, 0 com, 1 col.
4 - 2 dual, 2 com, 0 col.
5 - 2 dual, 1 com, 1 col.
6 - 2 dual, 0 com, 2 col.
7 - 1 dual, 2 com, 1 col.
8 - 1 dual, 1 com, 2 col.
9 - 1 dual, 0 com, 3 col.
10 - 0 dual, 2 com, 2 col.
11 - 0 dual, 1 com, 3 col.

4.2 Reliability Optimization

The analysis above compared architectures based only on the number of each type of spacecraft. No money was spent to improve the reliability of any components, implying the total reliability of the system could only be improved using redundancy. A model was introduced in Section 3.4.2 on page 69 to predict how much a component's reliability will increase if a given amount of money is spent on testing or improving the design. A logical next step in this process is to combine these two tools into one - a tool that can find the "best" architecture, in terms of all three life-cycle metrics, when an architecture is defined in terms of both the number of each type of spacecraft and the money spent to improve the reliability of individual components. The tool discussed here will once again only compare a given set of user-defined architectures. Please see Chapter 5 for a discussion of optimization tools used to explore the entire design space.

Once an architecture is defined in terms of the number of each type of spacecraft, additional money can be spent to improve the reliability of the individual components in the system. The user can define the amount of money to be spent on the entire system, including manufacturing spacecraft, operations, and improvements to component reliabilities, in one of two ways. Please see Section 3.3 on page 64 for a discussion of the model used to estimate this total system cost. The first method of defining the total amount of money spent is to define a total system budget. If this method is used, the cost per image metric becomes redundant with the number of images metric, since all systems will end up costing the same amount of money. The total system budget method is useful for projects that have a given budget, but have not yet decided on an architecture. This method allows these projects to find the most productive, reliable, and cost-effective systems possible within their budget. This method is shown in Equation 4.2, where *Budget* is user defined.

$$TotalSystemBudget = Budget \tag{4.2}$$

The total system budget method is not useful for projects in which the total budget is not set, but simply needs to be as low as possible. In this case, it may be advantageous to

examine systems that may not be as productive as other systems, but also cost significantly less money. The second method available to decide a total system budget for each architecture analyzed is to define the total system budget as a given percentage of the original cost of the system, without any improvements to components. With this method, a user can define a percentage, set at 20% as a default value, to be used. The total cost of each original system is then calculated. Next the final total system budget is calculated. For example, for the default percentage value, the initial system cost is multiplied by 1.2 to find the total system cost. This method of finding the total system cost is shown in Equation 4.3, where *percent* is user defined, and *InitialSystemCost* is found from the cost model with the number of each type of spacecraft defined from the user defined architecture and no money spent on improvements.

$$TotalSystemBudget = InitialSystemCost \times \left(1 + \frac{percent}{100}\right) \tag{4.3}$$

It should be noted that the sum of the money spent to improve each component does not necessarily equal the total system cost minus the initial system cost in either of these two methods. This implies that the concept of taking the amount of money defined by the total system budget minus the initial system cost and finding the optimum way of dividing this money to improve different components will not work. This is due to the operations part of the cost model. Since the operations cost is based on the number of baselines, a system that has had failures, and therefore has fewer baselines than it did originally, will cost less per month than the original system did. Therefore, if money is spent to improve the reliability of components, and therefore reduce the failure rates, the systems should remain in these more expensive initial states for longer than was originally the case. Therefore, since more money is needed for operations during the lifetime of the mission, the sum of the amount of money spent to improve the individual component reliabilities should be less than the total system budget minus the initial system cost.

4.2.1 Optimization Problem Formulation

The optimization problem to find the best way to spend money improving individual component reliabilities is formulated in Equation 4.4.

$$\begin{aligned} & Max[Rel(x,p)] \\ such\ that \quad & \begin{cases} Cost(x,p) < TotalSystemBudget \\ x_i > 0 \end{cases} \\ where \quad & \begin{cases} x = [Xmo\ Xlo\ Xb] \\ p = [dual\ com\ col\ \ldots] \end{cases} \end{aligned} \tag{4.4}$$

In Equation 4.4, x is the design vector consisting of the amount of money spent on improving the reliability of each component: combining optics (*Xmo*), collecting optics (*Xlo*), and bus (*Xb*). The parameter vector, p, includes the number of each type of spacecraft in the given architecture: dual functioning (*dual*), combining (*com*), and collecting (*col*). The parameter vector also includes a large number of other parameters that are held constant for all architectures (i.e. number of UV points needed, initial failure rates of components, etc.). *TotalSystemBudget* is the total system budget calculated by one of the two methods described above (see Equations 4.2 and 4.3).

The optimization problem shown in Equation 4.4 is solved using a simulated annealing algorithm. Please see Section 5.1.1 on page 105 for a description of simulated annealing. Each design variable is allowed to vary between zero and $100M in increments of $5M. In addition, since this optimization program needs to be run multiple times for each comparison study (once for each architecture being tested), it is important that it be time-efficient. Therefore, the algorithm is set to run 500 iterations before terminating and reduces the system temperature every iteration. Simulated annealing is a heuristic optimization technique, however, and is not guaranteed to find the true optimal solution. The probability of finding this true optimum also decreases as the number of iterations decreases. Therefore, a sanity check has been built into the optimization program to be certain that the solution reported has a possibility of being at or near the true optimal solution. While

there is no way to know before the optimization algorithm runs how any money should be divided among components to improve reliability, it is clear that if all of this money is not spent, then more money could be spent to improve one of the component's reliability, and therefore improve the entire system reliability. Therefore, once the simulated annealing algorithm has returned an initial solution of how much money should be spent to improve each type of component's reliability, the optimization algorithm checks the total cost of the system reported, including spacecraft costs and improvement costs. If the total system cost of the architecture reported by the simulated annealing algorithm is within 98% of the allowable system budget, calculated either as a user input or as the initial system cost plus a user defined percentage of the initial system cost, the solution is accepted and returned as the optimal solution. If the solution the simulated annealing algorithm returns produces a total system cost of less than 98% of this allowable system budget however, the optimization algorithm does not accept the solution and the simulated annealing algorithm is called again. The initial starting point for this second call to the simulated annealing algorithm is set at the final solution returned by the first call to the algorithm. This process is repeated until the optimization algorithm accepts the solution returned by the simulated annealing algorithm, and therefore returns the solution as an "optimal" solution.

4.2.2 Results

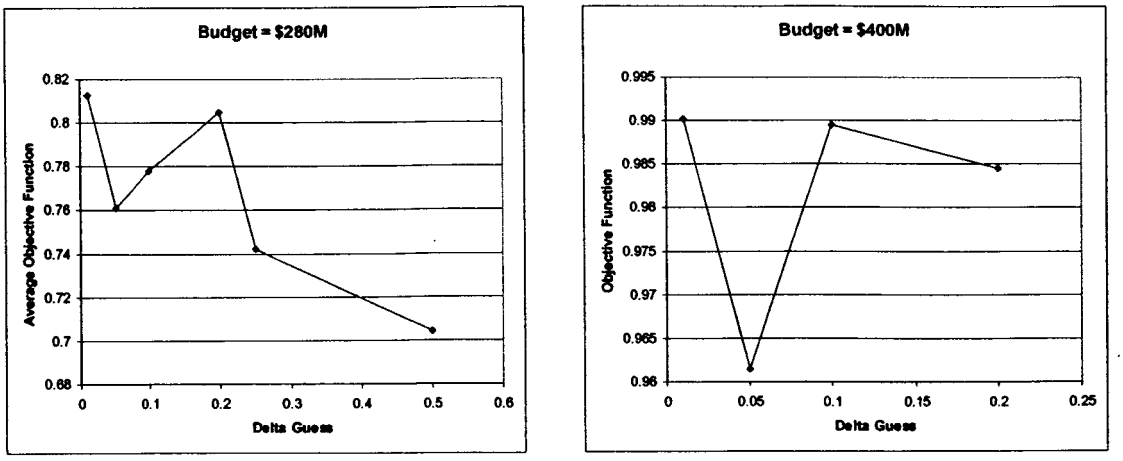
User defined architectures can now be analyzed and compared when additional money was allowed to be spent to improve the reliability of the components. Input architectures are again defined as the number of each type of spacecraft. In addition, the user defines which method, total system budget or percent of initial cost, is used for determining the total cost of the system. These methods are described above in Section 4.2. The user also defines any relevant parameters to either method, such as the total budget or the percent of the initial cost. The output architectures are defined as the number of each type of spacecraft and the money spent to improve the reliability of each component: combining optics, collecting optics, and bus. The architectures are compared using the four metrics

described in Section 4.1: productivity (number of images), reliability, cost-effectiveness (cost per image), and combined "score".

The same case studies described in Section 4.1.2 on page 80 were carried out again, this time with extra money spent on improving component reliabilities. The case study of architectures with a total of four spacecraft was carried out with a total system budget of $280M enforced. The case study with a total of six spacecraft was carried out using twenty percent of the initial system cost to improve component reliabilities. The same weighting values as used in Section 4.1.2 were used for these case studies ($w_P = w_R = 0.3$, $w_C = 0.4$).

The first step in running this algorithm is tuning the simulated annealing parameters to try to achieve the best performance out of the optimization scheme. In this particular case, the only parameter needed to tune is the initial guess of the difference in reliabilities (objective functions) between two neighboring design vectors, *delta_guess*. The algorithm was run with different settings for this parameter. Each parameter setting was run ten times, with twenty iterations for each optimization. This was done for two different budget settings, $280M for the four total spacecraft with a total system budget of $280M, and $400M for the six total spacecraft using twenty percent of the initial cost of the system to improve the component reliabilities test case. The results can be seen in Figure 4.3. Since the objective function is a "larger is better" metric, a *delta_guess* of 0.01 returns the best average solution in both budget cases and is therefore used in both case studies described below.

The initial results, in terms of just the division of money to improve component reliabilities, for the case study of architectures with a total of six spacecraft can be seen in Figure 4.4. Please see Table 4.1 on page 81 for an architecture key for this case study. Due to the stochastic nature of the simulated annealing algorithm used to find the division of money, it is important that the first step in analyzing these results is for the designer, or user, to perform a sanity check and be sure that the optimizer has indeed found an optimal,



a. b.

Figure 4.3 Tuning data for simulated annealing algorithm to optimize distribution of money to improve different component reliabilities for a total system budget of a) $280M and b)$400M.

or at least near optimal, method of dividing the money for each architecture. This is accomplished most easily by finding patterns, or trends, to the division of money and ensuring that the solution for each architecture follows these trends. The trends for this case study are seen in Table 4.3 and Table 4.4.

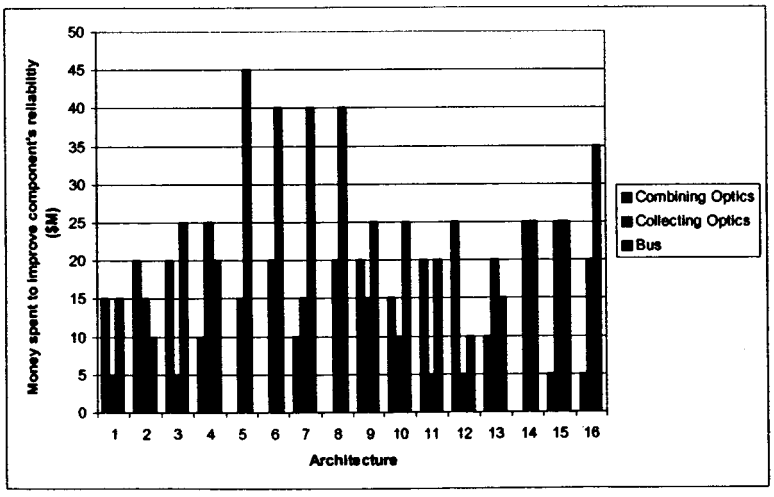


Figure 4.4 Initial results of division of money to improve component reliabilities for the case study with architectures with a total of six spacecraft and 20% of the initial system cost being spent on improving component's reliabilities.

TABLE 4.3 Architectures from case study of combinations of six total spacecraft with 20% of initial system cost spent on improving component reliabilities, sorted in ascending order of the difference between the total number of spacecraft capable of collecting light minus the total number of spacecraft capable of combining light (*Diff.*). Highlighted architectures have more money spent on improving collecting optics than on improving combining optics.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus	Total col	Total com	Diff.
[illegible]									
[illegible]									
[illegible]									
[illegible]									
[illegible]									
[illegible]									
[illegible]									
[illegible]									
[illegible]									
3	2	1	3	20	5	25	5	3	2
9	4	0	2	20	15	25	6	4	2
2	1	1	4	20	15	10	5	2	3
10	3	0	3	15	10	25	6	3	3
1	0	1	5	15	5	15	5	1	4
11	2	0	4	20	5	20	6	2	4
12	1	0	5	25	5	10	6	1	5

In Table 4.3, the full architectures are listed, including the money spent to improve the reliability of each component, along with the total number of spacecraft capable of combining light, the total number of spacecraft capable of collecting light, and the difference between these last two categories. It should be noted that a dual functioning spacecraft is capable of both combining and collecting light and is therefore counted in both categories. This implies that while each architecture contains a total of six spacecraft, the sum of the total number of spacecraft capable of combining and collecting light will be greater than six as long as there are dual functioning spacecraft in the architecture. The architectures in Table 4.3 are sorted in ascending order of the difference between the total number of

spacecraft capable of collecting light minus the total number of spacecraft capable of combining light (*Diff.*). In addition, those architectures in which more money is spent on collecting optics than is spent on combining optics are highlighted. The trend in this case is that more money is spent on improving the collecting optics reliability than on improving the combining optics reliability only when the difference in the total number of spacecraft capable of collecting versus combining light is less than two. This makes intuitive sense since in order for an architecture to be functional two collecting spacecraft are required versus only one combining spacecraft. Once there are two or more additional spacecraft capable of collecting light than there are capable of combining light, the redundancy inherent in the architecture for the collecting spacecraft provides more reliability, and the combining optics reliability is improved first.

In Table 4.4, the full architectures are again listed, but are this time sorted in ascending order by the total number of dual functioning spacecraft in the architecture. In this table the architectures in which more money is spent to improve the reliability of one of the two types of optics than is spent to improve the reliability of the bus are highlighted. With the exception of architectures one and four, all architectures in which there are only one or no dual functioning spacecraft spend more money improving the reliability of one of the types of optics than improving the reliability of the bus. This also makes intuitive sense, since with both other types of spacecraft, combining and collecting, a failure in the optics or in the bus results in a failure in the spacecraft. Only in dual functioning spacecraft is a failure in the bus a worse case scenario than a failure of a set of optics. Therefore, it is hypothesized that this trend is indeed true, and that architectures one and four are not optimized correctly.

To test this hypothesis, architectures one and four are run again, this time with 1000 iterations of the simulated annealing algorithm instead of 500. The results of the original runs for these architectures are then replaced with these new results. The score metrics are computed again for all sixteen architectures once the new results are in place, since these score metrics depend on the maximum and minimum of each metric between all architec-

TABLE 4.4 Architectures from the case study of combinations of six total spacecraft with 20% of initial system cost spent on improving component reliabilities, sorted in ascending order of number of dual functioning spacecraft. Highlighted architectures have more money spent on improving the collecting or combining optics than on improving the bus.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus
1	0	1	5	15	5	15
2	1	1	4	20	15	10
12	1	0	5	25	5	10
13	1	2	3	10	20	15
3	2	1	3	20	5	25
11	2	0	4	20	5	20
14	2	2	2	0	25	25
4	3	1	2	10	25	20
10	3	0	3	15	10	25
15	3	2	1	5	25	25
5	4	1	1	0	15	45
9	4	0	2	20	15	25
16	4	2	0	5	20	35
6	5	1	0	0	20	40
8	5	0	1	0	20	40
7	6	0	0	10	15	40

tures being compared. The final set of results for this case study of combinations of six total spacecraft can be seen in Figure 4.5 and Tables 4.5 and 4.6. Tables 4.5 and 4.6 correspond directly with Tables 4.3 and 4.4, but with the new results for architectures one and four in place. The pattern in Table 4.5 is identical to that in Table 4.3, implying that the trend of spending more money on collecting optics rather than combining optics only when there are two or more additional spacecraft capable of collecting light than there are capable of combining light still holds. In addition, the pattern in Table 4.6 is more complete than in Table 4.4, implying that the hypothesis that more money should be spent on improving optics rather than the bus when only one or no dual functioning spacecraft are in the architecture is true.

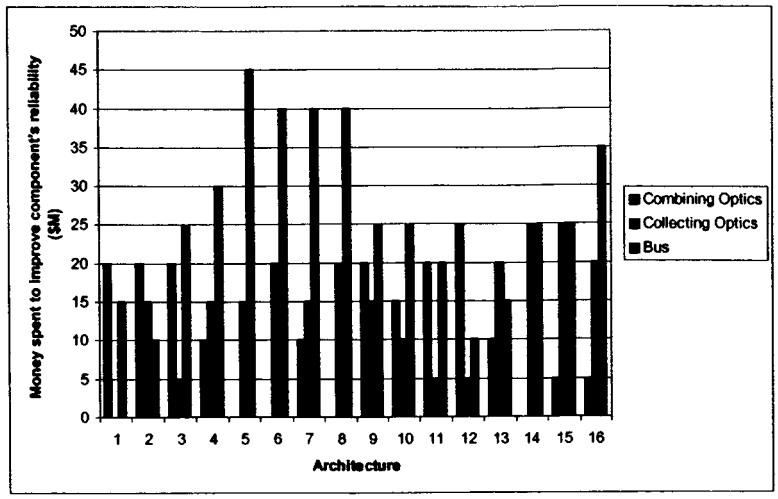


Figure 4.5 Final results of division of money to improve component's reliabilities for the case study with architectures with a total of six spacecraft and 20% of the initial system cost being spent on improving component's reliabilities.

While the optimization scheme did not find the true optimum for architectures one and four the first time around, it did find very good solutions. This can be seen in Figure 4.6, which shows the original "score" metric for all the architectures compared with the recalculated "score" metric once architectures one and four were replaced with better solutions. It is clear that the order of architectures from "best" to "worst" is preserved, and that the difference in all the architectures "scores" is very small. It should be noted that in general, changing two architectures design vectors should only change the "scores" associated with those two architectures. In this case, however, architecture one both before and after being re-run, has the lowest cost per image of any architecture. Therefore, when this architecture was re-run, and all of it's system metrics were slightly changed, it had an effect on all the architectures tested, and not just this particular architecture.

The final full results of this case study can be seen in Figure 4.7, which can be compared with Figure 4.1 to see the effect of using more money to improve the reliability of the components. While architecture seven still has the highest number of images and reliability and architecture one still has the lowest cost per image when additional money is spent to improve component reliabilities, the relative difference between architectures has

TABLE 4.5 Final architectures from case study of combinations of six total spacecraft with 20% of initial system cost spent on improving component reliabilities, sorted in ascending order of the difference between the total number of spacecraft capable of collecting light minus the total number of spacecraft capable of combining light (*Diff.*). Highlighted architectures have more money spent on improving collecting optics than on improving combining optics.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus	Total col	Total com	Diff.
3	2	1	3	20	5	25	5	3	2
9	4	0	2	20	15	25	6	4	2
2	1	1	4	20	15	10	5	2	3
10	3	0	3	15	10	25	6	3	3
1	0	1	5	20	0	15	5	1	4
11	2	0	4	20	5	20	6	2	4
12	1	0	5	25	5	10	6	1	5

changed. This is due to the fact that some architectures have larger changes to their productivity rate and reliability than other architectures when this extra money is spent. Therefore the architecture with the best combination of all metrics, or the best "score" metric, has changed from architecture ten (three dual functioning, no combining, and three collecting spacecraft) with no extra money spent, to architecture two (one dual functioning, one combining, and four collecting spacecraft) when extra money is spent to improve component reliabilities.

The initial results, in terms of just the division of money to improve component reliabilities, for the case study of architectures with a total of four spacecraft and a total system

TABLE 4.6 Final architectures from case study of combinations of six total spacecraft with 20% of initial system cost spent on improving component reliabilities, sorted in ascending order of number of dual functioning spacecraft. Highlighted architectures have more money spent on improving the collecting or the combining optics than on improving the bus.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus
1	0	1	5	20	0	15
2	1	1	4	20	15	10
12	1	0	5	25	5	10
13	1	2	3	10	20	15
3	2	1	3	20	5	25
11	2	0	4	20	5	20
14	2	2	2	0	25	25
4	3	1	2	15	10	30
10	3	0	3	15	10	25
15	3	2	1	5	25	25
5	4	1	1	0	15	45
9	4	0	2	20	15	25
16	4	2	0	5	20	35
6	5	1	0	0	20	40
8	5	0	1	0	20	40
7	6	0	0	10	15	40

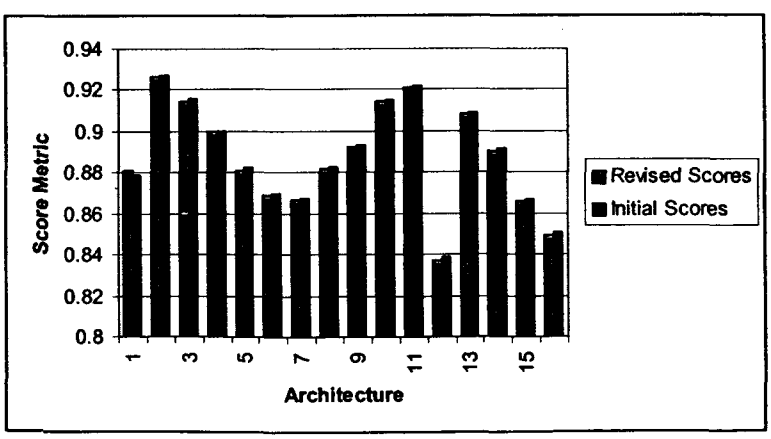


Figure 4.6 "Score" metrics for architectures in the case study of combinations of six total spacecraft with 20% of the initial system cost spent on improving component reliabilities both before and after architectures one and four were re-run.

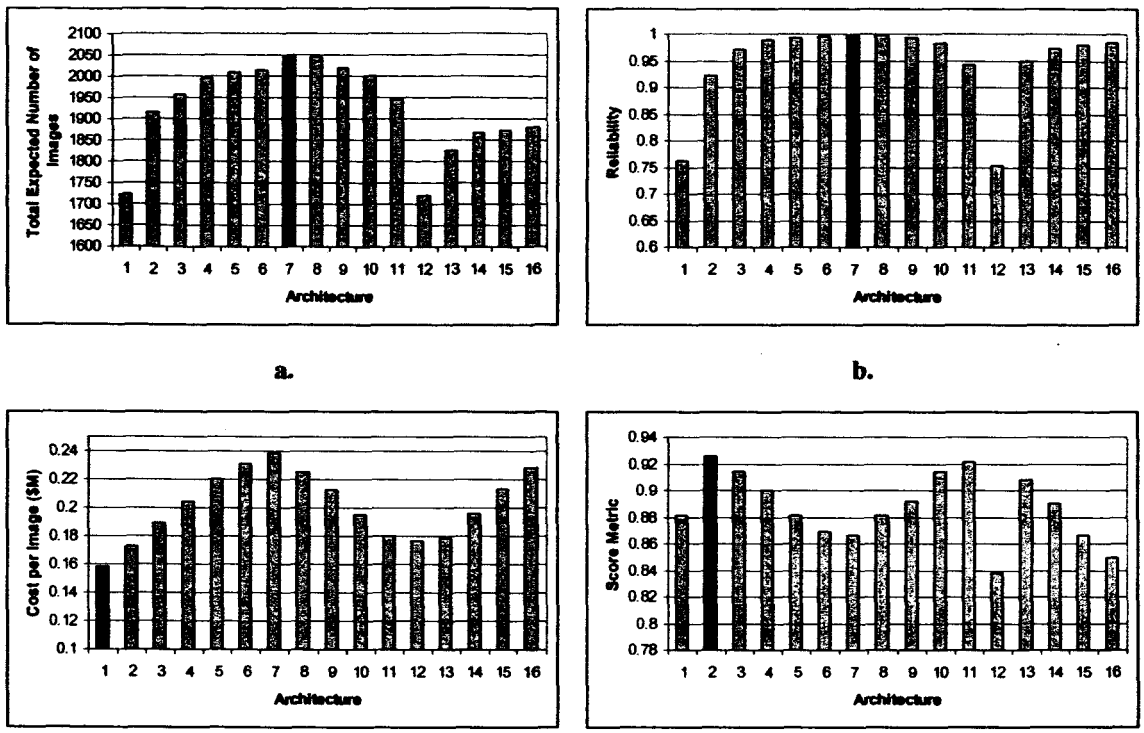


Figure 4.7 Final results for case study of combinations of six total spacecraft with 20% of initial system cost spent on improving component reliabilities. a) Expected total number of images for each architecture. b) Reliability for each architecture. c) Expected cost per image for each architecture. d) "Score" metrics for each architecture.

budget of $280M can be seen in Figure 4.8. Please see Table 4.2 on page 82 for an architecture key for this case study. Once again, the first step in analyzing these results is to perform a sanity check by finding patterns, or trends, to the division of money and ensuring that the solution for each architecture follows these trends. The trends for this case study are seen in Table 4.7 and Table 4.8.

Table 4.7 is the equivalent to Table 4.3 for this case study. The full architectures are listed, along with the total number of spacecraft capable of combining light, the total number of spacecraft capable of collecting light, and the difference between these last two categories. The architectures in Table 4.7 are again sorted in ascending order of the difference between the total number of spacecraft capable of collecting light minus the total number of spacecraft capable of combining light, and those architectures in which more money is

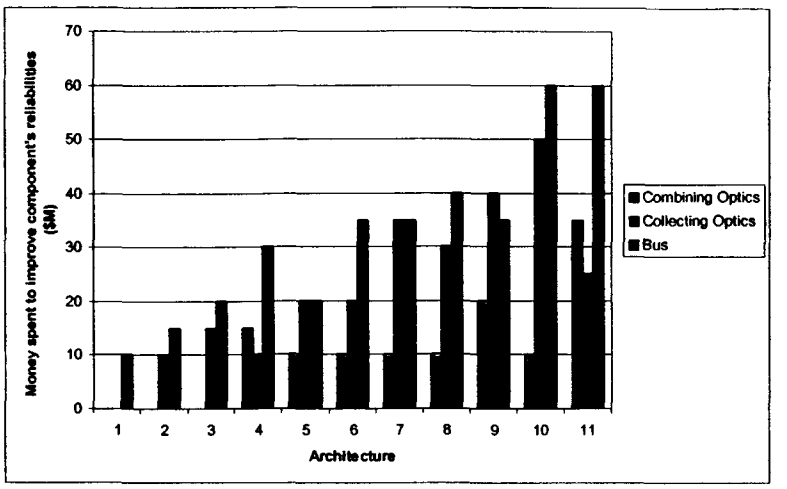


Figure 4.8 Initial results of division of money to improve component reliabilities for the case study with architectures with a total of four spacecraft and a total system budget of $280M.

spent on improving the collecting optics than is spent on improving the combining optics are highlighted. Since no trend is immediately evident in this case, it is hypothesized that the trend in this case is exactly the same as it was in the previous case study. More money should be spent on improving the collecting optics reliability than on improving the combining optics reliability only when there are two or more additional spacecraft capable of collecting light than there are capable of combining light. This hypothesis implies that three architectures should be run again - architecture four, architecture nine, and architecture six. Table 4.8 is the equivalent of Table 4.4 for this case study. The full architectures are again listed, but this time are sorted in ascending order by the total number of dual functioning spacecraft in the architecture. In this table the architectures in which more money is spent to improve the reliability of one of the two types of optics than is spent to improve the reliability of the bus are highlighted. There is only one architecture among the eleven tested in which the most money is spent to improve something other than the bus reliability. This implies a different hypothesis for when to spend more money to improve the reliability of the bus versus one of the sets of optics than was used in the first case study with more spacecraft. In this case, the clearest pattern, and therefore the natural hypothesis, is that the most money should always be spent to improve the bus reliability. If this hypothesis is correct, then once again architecture nine is not at the optimum point

for that combination of spacecraft and should be tested again. Therefore architectures four, six, and nine were all re-tested to try to find more optimal solutions.

TABLE 4.7 Architectures from case study of combinations of four total spacecraft with a total system budget of $280M, sorted in ascending order of the difference between the total number of spacecraft capable of collecting light minus the total number of spacecraft capable of combining light (*Diff.*). Highlighted architectures have more money spent on improving the collecting optics than on improving the combining optics.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus	Total col	Total com	Diff.
4	2	2	0	15	10	30	2	4	-2
11	0	1	3	35	25	60	3	1	2

The final set of results for this case study of combinations of four total spacecraft with a total system budget of $280M can be seen in Figure 4.9 and Table 4.9. Table 4.9 corresponds directly with Table 4.7, but with the new results for architectures four, six, and nine in place. Notice that while the distribution of money in architecture six is different than it was originally, it still exhibits the same pattern of the most money spent on improving the bus, followed by the collecting optics, followed by the combining optics. While this does not initially seem to fit within the original hypothesis of only spending more money on the collecting optics if the difference between the number of spacecraft capable of collecting light minus the number of spacecraft capable of combining light is less than two, it should be noted that the there are no combining spacecraft in this architecture. Therefore, one of

TABLE 4.8 Architectures from case study of combinations of four total spacecraft with a total system budget of $280M, sorted in ascending order of the number of dual functioning spacecraft. Highlighted architectures have more money spent on improving the collecting or combining optics than on improving the bus.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus
10	0	2	2	10	50	60
11	0	1	3	35	25	60
7	1	2	1	10	35	35
8	1	1	2	10	30	40
9	1	0	3	20	40	35
4	2	2	0	15	10	30
5	2	1	1	10	20	20
6	2	0	2	10	20	35
2	3	1	0	0	10	15
3	3	0	1	0	15	20
1	4	0	0	0	0	10

the two dual functioning spacecraft is required to be acting as a combining spacecraft at all times for the system to be functioning. This implies one less collecting spacecraft is available than was originally reported, making the difference between the total number of spacecraft capable of collecting light minus the total number of spacecraft capable of combining light equal to one. This explains why more money should be spent to improve the collecting optics of this system than to improve the combining optics. The other two architectures, four and nine, do have solutions which fall into the hypotheses stated above. Therefore, the hypothesis that more money is spent on combining optics only when there are two or more additional spacecraft capable of collecting light than there are capable of combining light holds for this case study as well. In this case study, however, it is never more advantageous to spend more money to improve either set of optics than to improve the bus. This trend is different in this case study than it was in the first case study, implying that all studies need to be examined individually and general trends of all systems cannot be drawn without first examining the systems in question.

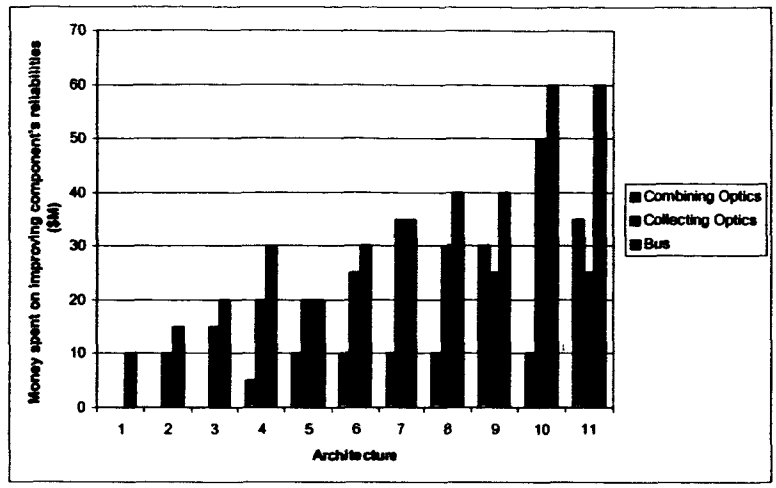


Figure 4.9 Final results of division of money to improve component reliabilities for the case study with architectures with a total of four spacecraft and a total system budget of $280M.

TABLE 4.9 Final architectures from case study of combinations of four total spacecraft with a total system budget of $280M, sorted in ascending order of the difference between the total number of spacecraft capable of collecting light mins the total number of spacecraft capable of combining light. Highlighted architectures have more money spent on improving collecting optics than on improving combining optics.

Arch.	Duals	Coms	Cols	$M on com optics	$M on col optics	$M on bus	Total col	Total com	Diff.
11	0	1	3	35	25	60	3	1	2
9	1	0	3	30	25	40	4	1	3

The final results from the four spacecraft, total system budget of $280M, case study can be seen in Figure 4.10, which can be compared to Figure 4.2 on page 82 from the case study

with no extra money spent to improve component reliabilities. While architecture eleven has the best overall combination of life cycle metrics, or "score", in both cases, it also has the highest number of expected images and reliability in this case study, compared to architecture one in the previous case study. This makes sense since architecture eleven had the lowest cost per image in the previous study, implying that it is the most cost-effective mission in terms of productivity. Therefore, it is not surprising that when the budget is forced to be the same for all architectures, the architecture that was the most cost effective is now the most productive.

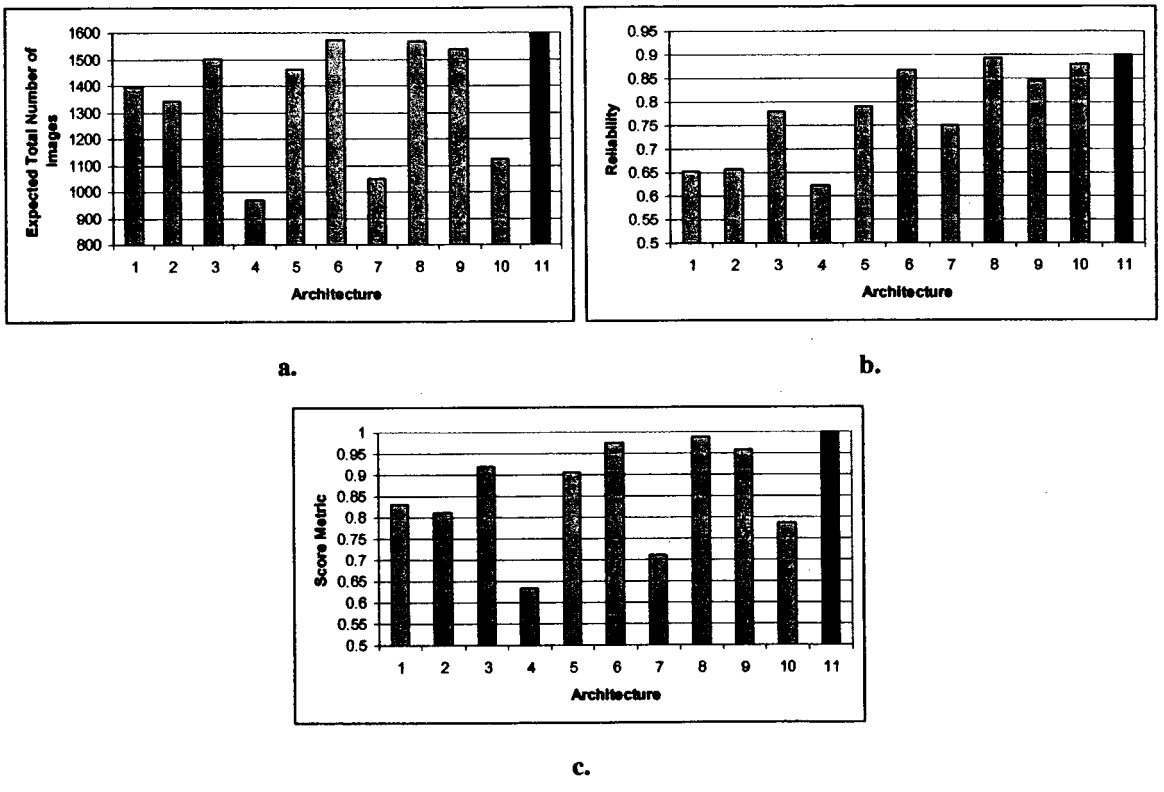


Figure 4.10 Final results for case study of combinations of four total spacecraft with a total system budget of $280M. a) Expected total number of images for each architecture. b) Reliability for each architecture. c) "Score" metrics for each architecture.

Figure 4.11 shows the "score" metric for each architecture in the four spacecraft case study both before and after architectures four, six, and nine were re-run. In a similar sense as with the six spacecraft case study, while the individual "scores" of the architectures that

were re-run change slightly, the overall pattern and trends remain consistent. Note that in this case, however, since none of the architectures which were re-run were the best in any metric category, the "scores" of those architectures that were not re-run remain constant. Both Figure 4.6 and Figure 4.11 show that while the simulated annealing algorithm may not always find the true global optimum, it does find a solution which is "good enough" to tell the patterns of which architectures are preferable over other architectures.

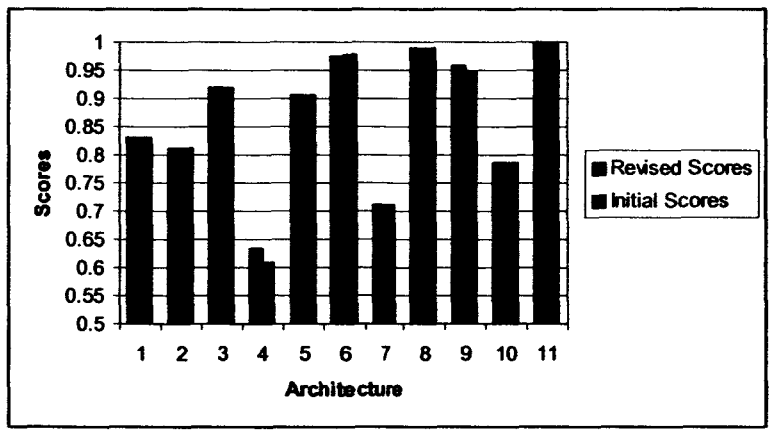


Figure 4.11 "Score" metrics for architectures in the case study of combinations of four total spacecraft with a total system budget of $280M both before and after architectures four, six, and nine were re-run.

4.3 Chapter Summary

This chapter has introduced a new metric designed to compare the total performance of different architectures. This metric has been tested, and results have been shown using two different case studies. It is worth noting that the architecture with the best total performance is not necessarily the architecture with the best performance in any of the three individual life-cycle metrics. This total performance metric was also tested, with the results shown in this chapter, for architectures which include extra money spent to improve component reliabilities. Through these case studies, trends were identified which give general rules of thumb on how to divide money among different components to improve individual reliabilities when trying to improve the overall system reliability as much as possible.

Chapter 5

SYSTEM OPTIMIZATION AND RESULTS

Once the tools are in place to analyze the productivity, reliability, and cost of individual architectures, the system with the "best" combination of these life cycle metrics can be identified. In Chapter 4 this process was described for comparing different user-defined architectures. This is an important process late in the conceptual design phase, when the designers have narrowed down the choices of architectures. This narrowing down process may have occurred any number of ways, such as only exploring designs which have aspects of heritage to them, or exploring the entire design space and deciding to look at only the most promising options. While the tools discussed in Chapter 4 can be used for this design space exploration by allowing one-at-a-time or orthogonal array design of experiments analyses, if the design space has a large number of degrees of freedom, each with many possible values, these types of analyses and experiments can be very lengthy and time consuming. Therefore, it is important to have tools that can effectively search the entire design space and provide insight into best areas of this design space and families of architectures to explore in further detail.

This chapter will explore three tools that are useful for design space exploration. Two heuristic optimization algorithms will be discussed - simulated annealing and genetic algorithms. Each of these algorithms will be introduced and applied to the SSI test case, with the results shown. Additionally, a sensitivity analysis tool will be discussed and applied to the results of the optimization techniques.

5.1 Heuristic Algorithms

Heuristic algorithms use rules of thumb to provide an intelligent, guided search through the entire design space. Heuristic algorithms differ from gradient, or analytic, optimization algorithms because there is no guarantee that the solution a heuristic algorithm has converged upon is the global, or even local, optimum. The solution reported by a heuristic algorithm is simply the "best" solution, in terms of the objective function, that the algorithm has found during its search of the design space. While analytic optimization techniques can guarantee a solution is at least a local optimum, these techniques often get stuck in these local optima and never find the global optimum. The design space for the problem of finding the optimal architecture, in terms of number of each type of spacecraft and amount of money spent to improve the reliability of each component, is very jagged. The number of each type of spacecraft is required to be an integer value and the set of feasible solutions is not necessarily continuous. Due to this jagged design space, the probability of any analytic optimization algorithm getting stuck in a local optimum is very high. In addition, heuristic algorithms have no constraints on linearity and do not require a convex design space [Jilla, 2002]. Finally, the goal of the optimization tools under development for this research is to find families of architectures, and trends in the best architectures, and not necessarily to find the architecture that is the global optimum in terms of the objective function. For instance, it would be very useful for a tool to provide two different architectures, each which have high productivity, low cost, and high reliability, but which lie in very different areas of the design space. These two architectures could then be compared in terms of other design measures which are more difficult to quantify, such as risk, political impact, and heritage.

Due to the ability to not get trapped in local optima, and the ability to find more than one "good" solution, heuristic algorithms were chosen to explore the design space and optimize the architecture in terms of the total productivity. Architectures are now defined as the number of each type of spacecraft and the money spent to improve the reliability of

each component. Two different heuristic algorithms were implemented: simulated annealing and genetic algorithms.

5.1.1 Simulated Annealing

The simulated annealing methodology is a heuristic optimization technique which applies statistical mechanics to optimization. This method was first introduced in 1983 by Kirkpatrick, Gelatt, and Vecchi [Kirkpatrick, Gelatt, and Vecchi, 1983]. Specifically, simulated annealing is an optimization method which mimics the cooling process of materials to a state of minimum energy. During the process of cooling from a liquid to a solid state, the molecules of a material will move around in a pseudo-random fashion. Each arrangement of molecules has a given energy. Lower energy states are preferred to higher energy states, and therefore the molecules continue to rearrange until a state of lowest energy is reached. If a material cools, or anneals, too quickly, it will solidify before the molecules have had time to arrange into this lowest energy state, and the material will be left in a sub-optimal energy state. If the material cools slow enough however, the molecules in the material will arrange in the correct order to form the lowest possible energy state. As the temperature is lowered, more of the material solidifies, and the molecules move around and rearrange less frequently, until the material is completely solidified and the molecules positions are locked in place.

This same process can be used to optimize a system in terms of a given objective function. The lowest energy state is equivalent to the optimal value of the objective function and the movement of molecules is equivalent to the evaluation of different neighboring design vectors. Two design vectors are considered neighbors if all but a given number of design variables are the same. The number of design variables allowed to vary for two vectors to still be considered neighbors is called the degrees of freedom of the optimization algorithm. The concept of neighboring design vectors is illustrated in Figure 5.1, where the three lower design vectors are all neighbors to the first design vector for two degrees of

freedom. For one degree of freedom however, only the second design vector is a neighbor to the first design vector.

[a b c d e f]

[b b c d e f]

[a g c d e k]

[a b a c e f]

Figure 5.1 Example of neighboring design vectors. The 3 lower design vectors are all neighbors to the top design vector with 2 degrees of freedom. Only the first and second design vectors are neighbors however if the degrees of freedom is set to 1.

As molecules are searching for the lowest energy state configuration when a material is cooling, they may accidently move to new configurations which are actually higher energy states than the original configuration. This is very likely to happen at the beginning of the cooling process when molecule movement is easier. As the material cools and the movement of molecules becomes more difficult, the probability of molecules rearranging themselves into a higher energy state decreases significantly. This process is mimicked in the optimization algorithm by what is known as the Metropolis Step [Metropolis et al., 1953]. The metropolis step is used to determine whether or not the system should move into a state with a worse objective function. Once a given design vector has been evaluated, a neighbor to that design vector is evaluated. If the neighbor design vector has a more optimal objective function value than the original design vector then the neighbor becomes the new starting point. A new neighbor is defined and the process is repeated. If the neighbor design vector has a less optimal objective function value than the original design vector, this neighbor may still become the new starting point with a probability determined by the Boltzman Factor. This factor takes into account both the temperature of the system, or how long the algorithm has been running, and the difference

between the two objective functions, and is defined in Equation 5.1 for a "larger is better" objective function. If the system temperature is low, implying that the algorithm is near completion, the system is less likely to move to a less optimal state. In addition, if the neighboring design vector is significantly worse than the original design vector in terms of the objective function, the system is also less likely to move to the neighboring state.

$$\Delta = J(x_o) - J(x_n)$$
$$BoltzmanFactor = e^{-\frac{\Delta}{T}} \tag{5.1}$$

In Equation 5.1, $J(x)$ is the objective function of design vector x, x_o is the initial design vector, x_n is the neighboring design vector, and T is the system temperature.

The Boltzman Factor and Metropolis Step keep the simulated annealing algorithm from getting trapped in local optima. They allow the algorithm to search in a worse direction every once in a while in order to be sure that this direction will not prove to be the better direction eventually. If the design space is thought of as peaks and valleys, representing local maxima and minima of the objective function, this step basically gives the algorithm a chance to look around the side of the peak or valley it is currently climbing to see if there is a bigger peak or lower valley elsewhere in the design space.

Before implementing a simulated annealing algorithm, several processes and variables need to be defined. First, the initial temperature needs to be defined such that the initial Boltzman Factor, or probability of moving to a worse design vector, is acceptable. This temperature depends both on what the user defines as an acceptable probability of moving to a worse design vector and on the potential, or average, difference between two neighboring design vector's objective function values. Next, the cooling schedule for the algorithm needs to be defined. This schedule sets how the temperature is changed in every iteration. The temperature should decrease as the algorithm proceeds, however the rate and method (exponential decrease, subtraction, constant factor, etc.) of this decrease can be different for every problem. A termination criteria can then be set. This criteria is usu-

ally either when the temperature reaches a given value near zero, or when a given number of iterations have been completed. The steps involved in setting up and implementing a simulated annealing optimization algorithm are shown below (for a "larger is better" objective function) [Jilla, 2002].

1. Define initial temperature of the system.
2. Define the cooling schedule.
3. Define the termination criteria (temperature equals zero, or number of iterations reached)
4. Run the algorithm
   - 4.1 Evaluate objective function at current location, $J(x_o)$.
   - 4.2 Find neighboring design vector (within designated degrees of freedom).
   - 4.3 Evaluate objective function at neighboring location, $J(x_n)$.
   - 4.4 If $J(x_n) > J(x_o)$, make x_n new current location.
   - 4.5 If $J(x_n) < J(x_o)$, make x_n new current location with probability $e^{(-\Delta/T)}$, where $\Delta = J(x_o)-J(x_n)$ and T = system temperature.
   - 4.6 Reduce temperature using cooling schedule.
   - 4.7 Repeat until termination criteria is met.

SSI Design Implementation

A simulated annealing algorithm is implemented with a Matlab script called "*sim_annealing.m.*" The first step in implementing a simulated annealing algorithm to find the "optimal" architecture for a separated spacecraft interferometry system is to define the objective function. The objective function is shown in Equation 5.2 below, and is very similar to the "score" metric described in Section 4.1 on page 76. In the case of the objective function however, the entire design space is being sampled and the objective function needs to be evaluated before all other architectures have been evaluated. Therefore, in the equation for the objective function the maximum of each metric is replaced by a user defined average value, found from experience from running other cases. Equation 5.3 shows an example objective function with typical values for the user defined inputs.

$$J(x) = w_R \frac{Reliability(x)}{AvgReliability} + w_P \frac{Productivity(x)}{AvgProductivity} + w_C \frac{AvgCostPerImage}{CostPerImage(x)} \quad (5.2)$$

$$J(x) = 0.3 \frac{Reliability(x)}{0.8} + 0.3 \frac{Productivity(x)}{1250} + 0.4 \frac{0.2}{CostPerImage(x)} \quad (5.3)$$

In Equations 5.2 and 5.3, *J(x)*, *Reliability(x)*, *Productivity(x)*, and *CostPerImage(x)* are the objective function, reliability, productivity, and cost per image respectively, evaluated at design vector x. In addition, w_R, w_P and w_C are the weighting factors applied to reliability, productivity, and cost, and *AvgReliability*, *AvgProductivity*, and *AvgCostPerImage* are the user defined average, or normal, values for the reliability, productivity, and cost per image respectively.

The next step in implementing a simulated annealing algorithm is to define both an initial temperature and a cooling schedule. The initial system temperature is set based on a user-defined input of a reasonable estimate of the difference between two neighboring architecture's objective functions. This value would again need to come from the experience of running other cases. The initial temperature is then set such that the initial probability of the algorithm defining a "worse" architecture as the new initial architecture is approximately 0.75. This can be seen in Equation 5.4, where Δ_{guess} is the estimate of the difference between two neighboring vectors' objective functions and T_o is the initial system temperature.

$$T_o = ceil\left(\frac{-\Delta_{guess}}{\ln 0.75}\right) \quad (5.4)$$

The cooling schedule is again user controlled. The user defines both the total number of iterations and the number of steps down in temperature desired. In order to get a low probability of jumping to a worse architecture at the end of the algorithm, the temperature should approach zero. Therefore the final temperature is set to be a value near zero, 0.001. The cooling schedule is set such that the temperature is reduced by the same factor in every step. This factor is found by using the number of steps required to lower the temper-

ature and the initial and final temperatures. In addition, the algorithm also calculates how many iterations should be run at each temperature step. This process is shown in Equations 5.5 through 5.7.

$$RunsPerStep = \frac{iterations}{steps} \tag{5.5}$$

$$F_{reduce} = \left(\frac{0.001}{T_o}\right)^{\frac{1}{steps}} \tag{5.6}$$

$$T_n = F_{reduce} \times T_{n-1} \tag{5.7}$$

In Equations 5.5 through 5.7, *iterations* is the number of iterations, *steps* is the number of steps down in temperature, T_o is the initial system temperature, F_{reduce} is the factor that the temperature is reduced by each step down in temperature, and T_n is the temperature at step n.

As the algorithm runs, the data from each analysis, including the objective function, temperature, and design vector, is stored in a matrix called *data*. The same data, but only for the design vectors which are redefined as the current design vector, is stored in a matrix called *data_proceed*. The final "optimal" design vector is the found by finding the maximum objective function stored in the matrix *data*.

It is clear that the solution returned by this algorithm is dependant upon several parameters. These parameters include the number of iterations, the number of steps down in temperature, and the initial guess at the difference in the objective function of two neighboring design vectors. Therefore, it is a good idea to "tune" the algorithm before it is used to find an "optimal" solution. This is accomplished by running the algorithm multiple times, with a small number of iterations each time, and varying each of the parameters for each run. Each parameter should be tested at a low, nominal, and high level. The final "optimal" objective function value for each setting of each parameter can then be compared to find the best setting of each parameter for the particular total system budget being

run. Due to the stochastic nature of the algorithm, if time permits, it is a good idea to run each parameter at each setting multiple times and use an average of the results to compare parameter settings. In general, the number of iterations should be as high as possible, and therefore this parameter does not generally need to be tuned. Instead, a small number of iterations is used for the tuning process in order to make the process run quicker. The number of steps down in temperature can be adjusted to the number of iterations (i.e. try one step down every five iterations, every two iterations, and every iteration, instead of every 200, 500, and 1000 iterations). These "optimum" settings for the parameters may change with the total system budget, and therefore this process of tuning parameters should be repeated before running a larger optimizing case for every major change in budget.

Due to the nature of design work and design problems, the "optimal" solution is not always what the user is the most interested in. Often times, the user is looking for as many solutions as he/she can find that are "good", and not necessarily one "best" solution. This allows the user to then identify trends among "good" solutions as well as to further examine these solutions in terms of other parameters not captured in this analysis, such as risk, political effect, and heritage. For this reason, two additional matrices are returned from the simulated annealing code, "*sim_annealing.m*." The first matrix, *good_obj_func*, returns all design vectors that have objective functions equal to 99% or higher of the maximum objective function found. Since the differences in the objective function of these architectures are by definition very small, these architectures tend to vary only by the amount of money spent on each component to improve reliabilities, and not by the number of each type of spacecraft. Therefore, a second matrix, called *good_obj_func2*, is also returned. This matrix contains all the architectures which have a different number of at least one of the types of spacecraft from the "optimal" solution, but still have an objective function equal to 97.5% or higher of the "optimal" objective function. This matrix allows the designer to more completely explore the design space surrounding several unique, "good" architectures, instead of just one "best" architecture. This is particularly useful in early conceptual design stages when it is beneficial to identify multiple architectures to

carry forward into the next phase of the design process, and not just a single "optimal" point design.

Results

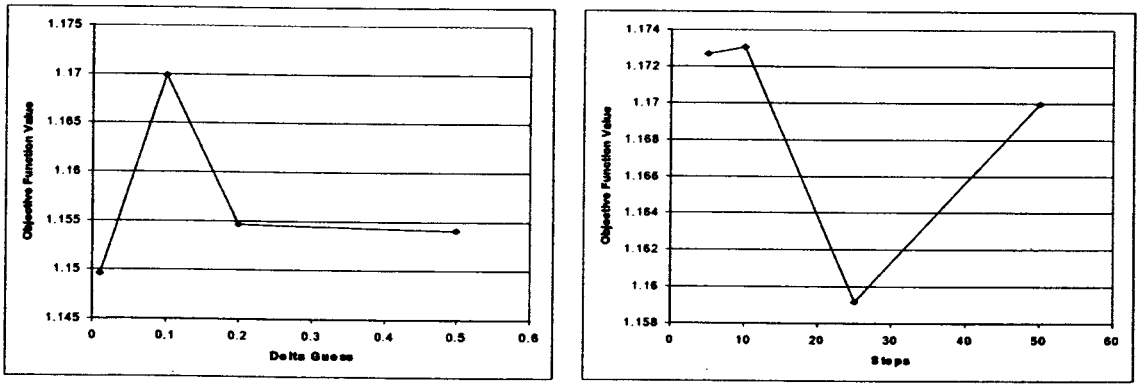
A case study was carried out using the simulated annealing algorithm described above. For this case study, the total system budget was set at $360M. The allowable bounds for each design variable were set as follows: zero to five dual functioning spacecraft in increments of one, zero to six combining spacecraft in increments of one, zero to nine collecting spacecraft in increments of one, and zero to one hundred million dollars spent on improving each of the component reliabilities (combining optics, collecting optics, or bus) in increments of five million. The bounds for the number of each type of spacecraft were set by estimating the highest possible number of each type of spacecraft that could be bought with the given budget. For example, the theoretical first unit cost of a dual functioning spacecraft is currently set at $70M. Since an architecture with solely dual functioning spacecraft is allowable, as long as there are at least three spacecraft, the maximum bound for dual functioning spacecraft is found by dividing the budget by the approximate cost of each spacecraft. This turns out to be slightly over five, implying that no more than five dual functioning spacecraft would be possible per architecture without breaking the budget constraint. The theoretical first unit costs for the combining and collecting spacecraft are currently set at $45M and $35M respectively. Since at least two collecting and one combining spacecraft are required for a functioning system, the maximum number of combining spacecraft is $360M minus two times $35M divided by $45M, which works out to slightly over six. Finally, the maximum number of collecting spacecraft is $360M minus $45M divided by $35M, which works out to exactly nine.

The first step in implementing the simulated annealing algorithm is to tune to the algorithm to find the most appropriate values for the initial guess at the difference between two neighbors' objective functions (*delta_guess)* and the number of steps down in temperature (*steps*). This was accomplished by testing each of these parameters at four different levels. For each test, the simulated annealing algorithm was terminated at 50 iterations. Each

test was repeated ten times to account for effects from the stochastic nature of the algorithm. A summary of the results can be seen in Table 5.1 and Figure 5.2.

TABLE 5.1 Simulated annealing algorithm tuning data.

Delta Guess	Steps	Average Objective Function
0.01	50	1.150
0.1	50	1.170
0.2	50	1.155
0.5	50	1.154
0.1	5	1.173
0.1	10	1.173
0.1	25	1.159
0.1	50	1.170



a. b.

Figure 5.2 Simulated annealing tuning data for a) Initial guess at difference in objective function between two neighbors and b) number of steps down in temperature.

This tuning data shows a maximum in the objective function at an initial guess of the difference between two neighbor's objective functions of 0.1 and ten steps down in temperature, and these values were therefore used for the future runs of the simulated annealing algorithm for this total system budget ($360). It should be noted that while ten steps down

in temperature was the maximum value, this was with only 50 total iterations. Therefore, future runs were set to one step down in temperature for every five iterations.

The results of this case study with 1,500 iterations can be seen in Figure 5.3 and Tables 5.2 and 5.3. Figure 5.3 shows the objective function value of the design vector that the algorithm chose as the new "current solution" for each iteration. Note that in the first iterations the algorithm chose a "worst" solution relatively frequently. This is due to the higher temperature at these iterations affecting the Boltzman factor. As the iteration number increases, and the system "cools down", this jump to a worse solution trails off, and the algorithm converges on a solution.

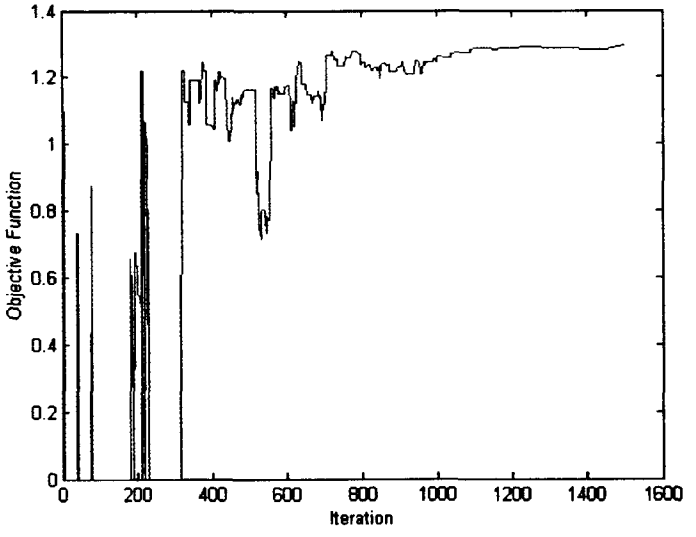


Figure 5.3 Simulated annealing convergence data

Table 5.2 shows the solution that the simulated annealing algorithm converged to. Note that no duals were used, and that more money was spent on improving the reliability of the combining optics than on improving the reliability of the collecting optics, which is consistent with the rules of thumb found in Section 4.2.2 on page 86 for how to spend extra money on improving components. Dual functioning spacecraft cost more than combining or collecting spacecraft alone, therefore it is possible that using more of the simpler,

cheaper spacecraft is more effective than fewer of the more complex, more expensive spacecraft. This explains the zero dual functioning spacecraft solution.

TABLE 5.2 Simulated Annealing Results

"Best" Architecture
0 duals
2 combiners
5 collectors
$10M on combining optics
$5M on collecting optics
$25M on bus
Objective Function = 1.2937

Finally, Table 5.3 shows the other "good" architectures returned by the algorithm. The first architecture in the table is the "best" architecture, followed by a second architecture, which has a different combination of spacecraft from the "optimal" solution but still has an objective function within 97.5% of the maximum objective function. These were the only two combinations of spacecraft found that came within 97.5% of the maximum objective function. This is most likely due in part to the algorithm's nature of following one path through the design space, leading to less possibility of finding completely separate architectures with equal, or near equal, performance. In fact, the only other architecture found with a different number of spacecraft that performed near the "best" architecture is still very similar to the "best" architecture in that no dual functioning spacecraft are used and instead two combining spacecraft provide redundancy for the combining optics. The last four architectures listed in Table 5.3 have objective function values within 99% of the "best" architecture, but have the same core combination of spacecraft and simply differ from the "best" architecture by the division of extra money to improve component reliabilities. They are still useful to report however, both to see what combinations the algorithm has tried that have come out worse than the "best" architecture, and for future exploration of the design space. It should be noted that the last four rows of

Table 5.3 are simply a sampling of the architectures returned that fell into this category since 30 different architectures were returned.

TABLE 5.3 Good architectures returned from simulated annealing. The first two are within 97.5% of the "best" architecture's objective function and vary by the number of each type of spacecraft (the first architecture listed is the "best" architecture), while the last 4 are within 99% of the "best" architecture's objective function but only vary by the money spent to improve different component reliabilities.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2937	0	2	5	10	5	25
1.2876	0	2	4	10	20	25
1.2909	0	2	5	10	15	35
1.2903	0	2	5	10	20	25
1.2892	0	2	5	10	0	30
1.2890	0	2	5	15	5	40

5.1.2 Genetic Algorithms

Genetic algorithms are a set of heuristic algorithms based on Darwin's principle of survival of the fittest first proposed by J. Holland of the University of Michigan in 1975 [Holland, 1975]. The principle of survival of the fittest says that of a generation of varied individuals, only those who have the traits necessary to survive will live long enough to mate and have offspring. In this way, the individuals with traits that are not advantageous to survival will be killed off before reproducing, and these traits will essentially vanish. In addition, the more fit or better suited an individual is to survival, the more other individuals will want to mate with it, and it's traits will be spread farther. Therefore, after several generations of evolution, the later generations in this society will only contain the advantageous traits and the individuals in these later generations should be stronger and more fit for survival than any of their ancestors in the initial population.

This concept of survival of the fittest is translated to an optimization algorithm by the analogy that individuals are particular design vectors and the fitness of a given individual is the value of the objective function for that design vector. A random initial set, or population, of design vectors is chosen to begin the algorithm. Each design vector in this population is then encoded to form a "gene" for that design vector. This step is needed in order to model mating and mutations of individuals. There are several ways to encode design vectors to "genes", including binary and float encoding. In these types of encoding each design variable is changed to a binary or float number representation, respectively. If the design variables are continuous numerical values this encoding process is simple and can be just the variable value itself. If the design variables are non-continuous or not numeric this encoding process is more complicated.

Once the design vectors are encoded as "genes" they can mate and mutate to form a new population. The probability of an individual design vector being able to "mate" with another design vector is determined by it's fitness, or objective function. The more optimal the objective function, the higher the probability that the individual will mate. There are several ways to choose individuals to mate, including ranking schemes, fitness value schemes, and roulette wheel selection. Ranking schemes involve ranking the entire population in order from the most fit to the least fit. An individual's probability of mating is then equal to the inverse of the ranking. In fitness value schemes, the probability of an individual mating is equal to the value of the individual's fitness function over the sum of the values of the fitness functions for all individuals in the population. In roulette wheel selection, an individual is given a unique range of values, with the range depending on the fitness of the individual. The more fit the individual, the larger the range of values. Next, a random number is generated that must fall into one of the ranges of values of the individuals. If the random number falls into a given individual's range of values, then that individual is chosen to mate [deWeck, 2002]. The roulette wheel scheme can be combined with fitness value schemes if the probability of choosing the individual, or the range of values of that individual, is calculated using a fitness value scheme. With tournament selection, the algorithm selects a given number of individuals, with replacement, ran-

domly. From this set of individuals, the individual with the best fitness function is chosen to mate. This process is repeated, with a new set of randomly selected individuals, until the correct number of "parents" are chosen [Houck, Joines, and Kay, 1995].

Once two individuals are chosen to mate, their genes, or encoded design vectors, must be combined to create "children." This process is called crossover. Crossover can produce anywhere from one to many children from two parents. In single point crossover, a random point along the genes, or encoded design vectors, is chosen. All the encoded material from one side of this point from one parent is put together with all the encoded material from the other side of the other parent to create one encoded gene. In path relinking, each child is a neighbor of either a parent or another child. For example, child one is a neighbor to parent one, child two is a neighbor to child one, child three is a neighbor to child two, and so on to child n, who is a neighbor to parent two. Two design vectors are considered neighbors if all but a given number of design variables are the same, as shown in Figure 5.1. Whichever way crossover is achieved, a new population is created using the information from the initial population.

Once a new population is created, a process known as mutation needs to occur. Mutation allows for stochastic processes to be included in the algorithm and restore diversity to the population. Mutation accounts for a small probability that any piece of an encoded design vector is changed after a child is created. For example, in binary encoding, the mutation probability, or rate, would be the probability that any bit in any design vector is changed from a zero to a one after a child is created. This mutation is necessary to keep the populations from becoming static too early, and increases the probability of exploring the entire design space.

Once a new generation is created and then mutated, the process repeats itself, with the analysis of all individuals in the new generation. This process is continued until a termination criteria is met. The termination criteria can be a tolerance on the diversity of a population, or a set number of generations completed. A set number of generations is a very

common termination criteria for genetic algorithms. The analogies between natural selection and genetic algorithms are summarized in Table 5.4, and the steps involved in implementing a genetic algorithm are given below [deWeck, 2002].

1. Define objective, or fitness, function.
2. Define mating selection criteria and crossover rate.
3. Define mutation rate.
4. Initialize algorithm.
   4.1 Initialize first random population.
   4.2 Evaluate fitness of all individuals.
   4.3 Select individuals to mate.
   4.4 Create new generation.
   4.5 Mutate new generation.
   4.6 Repeat from Step 4.2 until termination criteria is met.

TABLE 5.4 Analogies between natural selection and genetic algorithms

Natural selection	Genetic algorithms
Individuals	Design vectors
Genes	Encoded design vectors
Generations	Groups of design vectors
Population size	Number of design vectors per generation
Fitness	Objective function
Mating	Combining design vectors (crossover)
Children	New design vectors (combinations of old design vectors)
Mutation	Random changes in encoded design vector

SSI Implementation

Genetic algorithms were implemented for the separated spacecraft interferometry system design problem by using the publicly accessible Matlab toolbox *GAOT*. The same objective function that was used for the simulated annealing algorithm was also used as the genetic algorithm fitness function. Please see Equations 5.2 and 5.3 for this fitness func-

tion. The initial population is chosen randomly from the set of possible individuals defined by the user given bounds on each design variable. This population was chosen using the toolbox function "*initializega.m.*" The first three design variables contain integer constraints (one cannot buy half a spacecraft) and therefore, to accommodate this constraint, a modified binary scheme was chosen for the encoding of the design vectors to "genes." Several modifications were necessary in order to get the binary encoding aspects of toolbox algorithm working correctly. Additional modifications were made to this binary encoding to ensure the integer constraints needed for this problem were met. Specifically, a line was added which rounded the float number to the nearest integer after changing from binary to float formats. Once these modifications were made, they were tested by running several iterations through a function which chose a random integer design vector, encoded it using the binary encoding scheme, decoded it, and checked the original vector against the new decoded vector. These tests led to confidence in the modified encoding scheme. The full list of modifications to the *GAOT* toolbox can be seen in Appendix B. It should be noted that float encoding schemes are also included in the *GAOT* toolbox.

The *GAOT* toolbox offers three different selection criteria functions: roulette wheel, normalized geometric ranking, and tournament. The roulette wheel selection criteria is implemented using a fitness value scheme to find the probability of choosing each individual. The normalized geometric ranking scheme is a modified ranking scheme. Please see Houck, Joines, and Kay for more information on this scheme [Houck, Joines, and Kay, 1995]. For this research, the tournament selection scheme was used. Please see above for a description of the tournament scheme. Once a new population is chosen, crossover, or mating, and mutation occur at a user-defined rate, or probability. Several options are available for schemes of how to accomplish crossover and mutation. For this research, the default values of simple, or single point, crossover, and binary mutation, in which each bit in each individual is flipped with a given user-defined probability, or mutation rate, were chosen [Houck, Joines, and Kay, 1995].

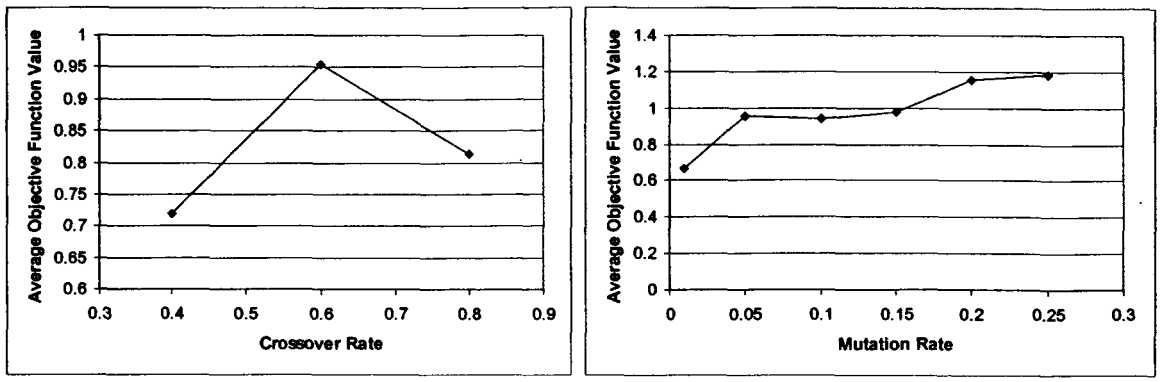
Once again, since the true "optimal" solution is not necessarily of interest to the designer in this problem, but rather a list of "good" solutions is, all the data from all analyses executed during a genetic algorithm scheme are saved in a matrix called *data*. This data can then be examined to find all solutions which produce objective function values within given ranges of the "optimal" solution. Solutions with objective function values within 99% of the "optimal" solution are reported, along with solutions within 97.5% of the "optimal" objective function value, but with a restriction that the number of at least one type of spacecraft must be different from the "optimal" solution.

Results

The *GAOT* toolbox was tested on a case study for a system with a total budget of $360M. This is the same case study that was carried out with the simulated annealing tool. The case study was carried out using 60 generations with a population of 50 individuals in each generation. Note that this leads to approximately 3000 analyses required to complete the study, which is equal to the approximate number of analyses required with the simulated annealing algorithm (1500 iterations, 2 analyses for each iteration). The design variables were all restricted to integer values with the number of dual functioning spacecraft ranging from zero to five, the number of combining spacecraft ranging from zero to six, the number of collecting spacecraft ranging from zero to twelve, and the money spent on improving each component's reliability ranging from zero to one-hundred. Please see Section 5.1.1 for the rationale behind these bounds.

The first step in implementing a genetic algorithm optimization scheme is to tune the algorithm parameters: crossover and mutation rates. This was accomplished by allowing the algorithm to run with fewer analyses at multiple levels for each parameter. Each test was repeated ten times and the average value of the objective function for the ten runs was used to compare the parameter level settings. The default crossover rate is 0.6 and was tested at 0.4, 0.6, and 0.8. The default mutation rate is 0.05 and was tested at 0.01, 0.05, and 0.1. This test led to a maximum objective function when the mutation rate was set at 0.1. Therefore, tests were completed for mutation rates at 0.15 through 0.25 as well, in

search of a maximum. The results for the tuning of the crossover rate and mutation rate are shown in Figure 5.4.



a. b.

Figure 5.4 Tuning data for genetic algorithms optimization scheme for a) crossover rate and b) mutation rate.

While the average objective function is continuously increasing with the mutation rate, it is dangerous to allow this parameter to be too high. The increased objective function for higher mutation rates may have been a response to the lower number of individuals per generation, leading to a higher probability of finding no feasible solutions in any given generation. With a higher mutation rate, the search is more random and the algorithm has a higher probability of testing entirely new individuals each generation, leading to a higher probability of finding at least one feasible individual in the short tuning tests. This problem would be avoided with a larger population size per generation however, and the higher mutation rate would simply make the search more of a random search than a guided search through the design space. Therefore it was decided to run the full genetic algorithm optimization scheme at mutation rates of 0.05, 0.1, and 0.15. The default value of 0.6 for the crossover rate produced the maximum objective function value, and was therefore used throughout the rest of the tests discussed here.

The results for the genetic algorithm optimization scheme for a total system budget of $360M with a mutation rate of 0.1 are shown in Tables 5.5 and 5.6 and Figure 5.5.

Table 5.5 shows the "optimal" solution reported by the genetic algorithm. Comparing Table 5.5 and Table 5.2 on page 115 show that the two solutions, found with genetic algorithms and simulated annealing respectively, are very similar. Both solutions have two combiners, five collectors, and no dual functioning spacecraft. In addition, both solutions spend more money on improving combining optics than on improving collecting optics, which is also supported by the rules of thumb developed in Section 4.2.2, since this architecture has three more spacecraft capable of collecting light than capable of combining light. Both solutions also spend $25M on improving the reliability of the bus. The genetic algorithm obtains a slightly better solution by spending more money on improving both the combining and collecting optics' reliabilities than the simulated annealing solution does. The similarity in the two solutions, however, provides greater confidence in both.

TABLE 5.5 Genetic algorithm results (mutation rate = 0.1)

"Best" Architecture
0 duals
2 combiners
5 collectors
$12M on combining optics
$9M on collecting optics
$25M on bus
Objective Function = 1.2949

Table 5.6 shows the other architectures returned by the genetic algorithm that are close to the "optimal" solution. The first row in Table 5.6 is the "optimal" solution reported. The following rows are the other architectures found which have an objective function value within 97.5% of the "optimal" solution while still having a different number of at least one type of spacecraft. Note that seven of these additional architectures were found with the genetic algorithm, compared to one additional architecture found with simulated annealing. While not shown in Table 5.6, the genetic algorithm optimization scheme also returned over 240 additional solutions that have objective function values within 99% of

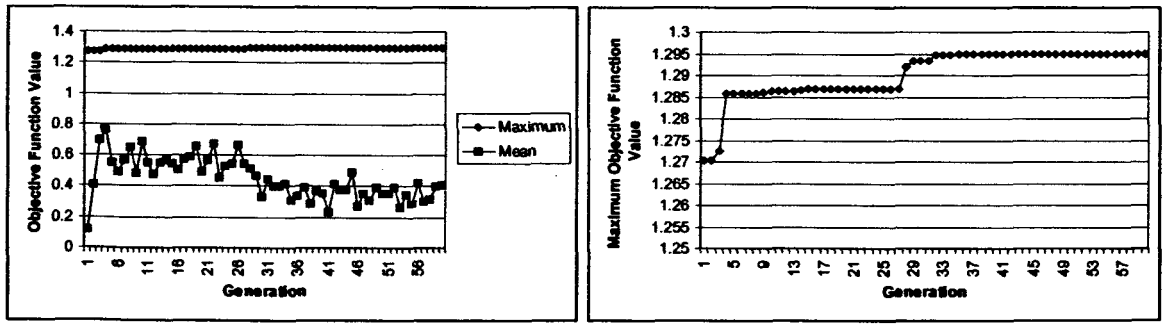
the "optimal" solution, but only vary from the "optimal" solution by the division of money spent to improve reliabilities. In many cases and stages of design, solutions such as the ones reported in this case study are often much more valuable to the designer than the one true "optimal" solution. For these cases, it may prove beneficial to use the genetic algorithm optimization scheme since it returned seven times more solutions in this category than the simulated annealing algorithm. The full listing of all solutions returned by the algorithm can be seen in Appendix C.

TABLE 5.6 Good architectures returned from genetic algorithms with mutation rate = 0.1. All architectures listed are within 97.5% of the "best" architecture's objective function and vary by the number of each type of spacecraft (the first architecture listed is the "best" architecture). The algorithm also returned over 240 architectures which are not shown here, with objective functions within 99% of the "best" architecture's objective function but only vary from the "optimal" solution by the money spent on different components.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2949	0	2	5	12	9	25
1.2868	1	1	4	12	13	30
1.2723	1	1	3	12	21	33
1.2702	0	3	5	3	3	25
1.2652	2	0	4	5	17	29
1.2650	2	1	3	9	10	18
1.2640	1	2	4	9	9	14
1.2632	0	2	6	1	3	19

Figure 5.5 shows the convergence history for the genetic algorithm solution with a mutation rate of 0.1. In Figure 5.5a, the blue points are the maximum objective function found in that generation while the red points are the mean objective function for the generation. The mean objective function is generally less than half that of the maximum objective function found for each generation. This is due to the jagged design space of the problem. If either the budget constraint or the operating constraint (at least two combining and one collecting spacecraft) is broken, the system is automatically assigned an objective function

value of zero. Therefore, a very slight change in one design variable will make the objective function of a design vector go from optimal, or near optimal, to zero. In addition, the upper bound of each design variable is set for the maximum possible while still meeting all requirements. This leads to a large number of combinations of design variable values that will be infeasible designs. This implies that generations will in general have some design vectors which are feasible and many design vectors which are infeasible. If only half the design vectors are infeasible and if all the other design vectors are optimal, the mean objective function value would be one half the maximum. Since in general more than half the solutions of a generation will be infeasible, and the majority of those that are feasible will have objective function values less than the optimal value, the mean value of the objective function for each generation is expected to be less than one half the maximum value. Figure 5.5b shows a closer view of the maximum objective function value for each generation. Notice that the maximum objective function is in general increasing as the generations increase, but is also converging to a final solution.



a. b.

Figure 5.5 Genetic algorithms (mutation rate = 0.1) convergence history - a) maximum and mean objective function for each generation, b) maximum objective function for each generation.

One of the most appealing aspects of using genetic algorithms for design problems is the ability to search multiple areas of the design space at once. This in general leads to a more thorough exploration of the entire design space than other optimization algorithms offer. Figure 5.6 shows two different views of a portion of the design space and the extent to which it was explored by the genetic algorithm utilized in this case study. Each blue dot in

Figure 5.6 is a different architecture, in terms of only the number of each type of spacecraft, which was analyzed at least once by the algorithm, and the red dot is the solution reported "optimal". While this only shows the design space covered by the first three design variables, due to the inability to visualize the full 6 degree of freedom design space, it is clear that the genetic algorithm explores almost the entire space. This is a very promising feature of using genetic algorithms for future space system conceptual design problems.

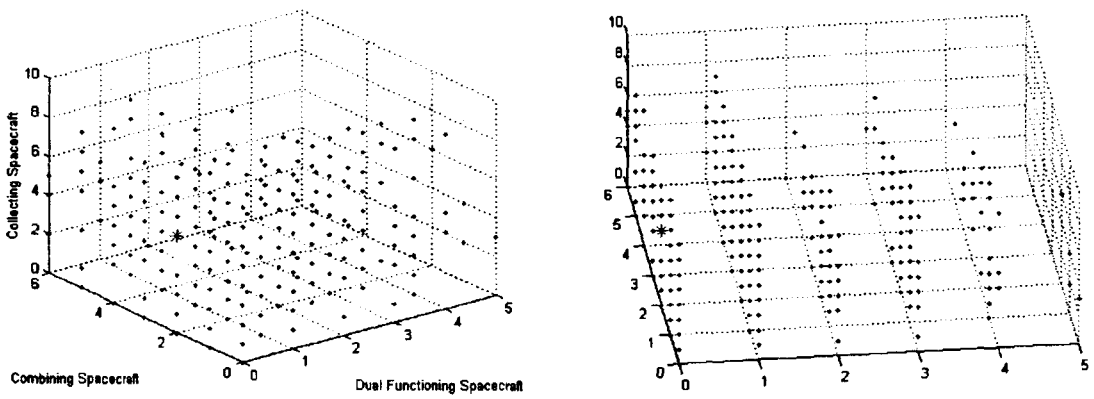


Figure 5.6 Design space exploration by genetic algorithms (mutation rate = 0.1). Red dot is solution reported as "optimal".

This same case study (total system budget of $360M) was repeated using a genetic algorithm optimization scheme with a mutation rate of 0.15 and 0.05. The crossover rate was maintained at 0.6, and binary encoding and mutation and tournament selection were still used. The results can be seen in Table 5.7, and Figure 5.7. Note that these studies converged to a solution with a lower objective function value than the previous case study (with mutation rate = 0.1). The case study with the mutation rate set at 0.1 also returned the largest number of solutions both within 99% of the "optimal" objective function solution and within 97.5% of the "optimal" objective function that differ from the "optimal" solution by the number of at least one type of spacecraft. Therefore it is assumed that a mutation rate of 0.1 is the optimal setting for this parameter. These full case study tests should be run if the tuning data for a particular parameter is in question, as was the case

for the mutation rate in this study. While these tests do require time to run, since a full case study is accomplished for each setting, they provide greater confidence in the final solutions returned when they are completed.

TABLE 5.7 Genetic algorithm results from three case studies for varying mutation rates. Architectures shown are "optimal" architectures returned by the algorithm for each mutation rate.

	Mutation Rate = 0.05	**Mutation Rate = 0.1**	**Mutation Rate = 0.15**
Dual Functioning Spc.	1	0	1
Combining Spc.	1	2	1
Collecting Spc.	4	5	4
$M on combing optics	14	12	14
$M on collecting optics	12	9	11
$M on bus	28	25	29
Objective Function	1.2872	1.2949	1.2871
Solutions returned within 99% of "optimal" solution	202	244	104
Solutions returned within 97.5% of "optimal" solution, with different number of at least one type of spc.	3	8	6

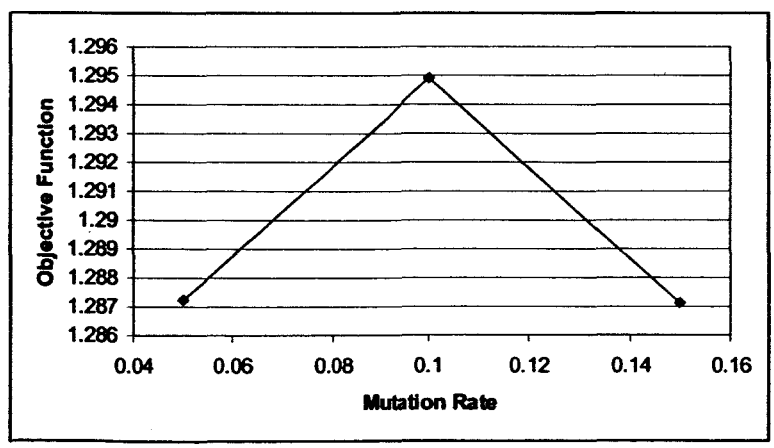


Figure 5.7 Mutation rate tuning information with full case studies.

5.2 Sensitivity Analysis

The analysis presented above is dependant upon 23 user defined inputs, or parameters, which are listed in Table 5.8. These parameters are constant for all architectures being evaluated, but can affect different architectures in different ways. The sensitivity of the solutions returned by all of the analyses discussed above to all of these parameters is crucial. Therefore a sensitivity analysis tool has been developed to find the sensitivity of the models to these parameters. This tool can be used to either check the robustness of the "optimal" or "best" architecture reported from one of the tools discussed previously, or to find the parameters which affect the outcome of the analyses and models the most. The parameters found to have the most effect on the outcome of the analyses can then be focused on starting early in the conceptual design stage to either improve the parameter value or decrease the uncertainty of the value.

The sensitivity of each life cycle metric to each parameter is calculated using finite differences. The parameter value is perturbed, and the resulting change in the life cycle metric is observed. This is shown in Equation 5.8, where Δp is the change in the parameter and Δm is the change in the life cycle metric (number of images, reliability, cost per image, or "score").

$$sensitivity = \frac{\Delta p}{\Delta m} \tag{5.8}$$

The number of perturbations (sensitivity points) and the percentage change between each perturbation (change in sensitivity points) is user defined. The total amount of perturbation is spread equally above and below the nominal parameter value. For example, if the user inputs three sensitivity points, with a change in sensitivity points of 0.01, the sensitivity code will do an analysis with each parameter set at 99%, 100%, and 101% of the nominal parameter value. In addition, the user controls which of the 23 possible parameters listed in Table 5.8 are examined. This gives the user flexibility over the detail in the sensitivity analysis versus the time required to run the sensitivity analysis code. One strategy a

TABLE 5.8 User defined parameters that affect the outcome of analysis results.

Variable Name	Description	Units
mo	Combining optics failure rate	months^{-1}
lo	Collecting optics failure rate	months^{-1}
m	Combiner bus failure rate	months^{-1}
l	Collector bus failure rate	months^{-1}
d	Dual functioning bus failure rate	months^{-1}
life	Mission design lifetime	months
S	Scale factor for increasing reliability with money spent	N/A
N	Number of pairs of UV points required per image	N/A
Co	Time required per pair of UV points	months
Ot	Overhead time per image	months
mo_tfu	Combiner optics theoretical first unit cost	$M
lo_tfu	Collector optics theoretical first unit cost	$M
mb_tfu	Combiner bus theoretical first unit cost	$M
lb_tfu	Collector bus theoretical first unit cost	$M
db_tfu	Dual functioning bus theoretical first unit cost	$M
ops	Operations cost per month per baseline	$M/month/ baseline
s_lc	Learning curve slope	N/A
w_cpi	Weighting of CpI in objective function	N/A
cpi_avg	Average value for CpI (used for scaling)	$M
w_noi	Weighting of NoI in objective function	N/A
noi_avg	Average value for NoI (used for scaling)	images
w_rel	Weighting of reliability in objective function	N/A
rel_avg	Average value for reliability (used for scaling)	N/A

user might employ is to take a small number of large step sizes first (i.e. 3 sensitivity points with a change in sensitivity of 0.05) looking at every parameter. Then, once the parameters which the metric in question is most sensitive to are identified, these parameters can be run through the sensitivity analysis again, with a greater number of smaller step sizes. This sensitivity analysis can be run on architectures defined only as the number of

each type of spacecraft (*"sensitivity.m"*), or on architectures with the division of money to improve reliabilities included in the design vector (*"sensitivity_full_x.m"*).

A case study was done on an architecture with zero dual functioning spacecraft, two combining spacecraft, and five collecting spacecraft without the division of money included in the design vector. This particular architecture was chosen since it was reported as the "optimal" design vector by both the simulated annealing and genetic algorithm optimization schemes. All parameters were examined at one percent intervals up to ±5% of the nominal design. The results from this case study can be seen in Figure 5.8. Only those variables that had any effect on the metric being studied are shown.

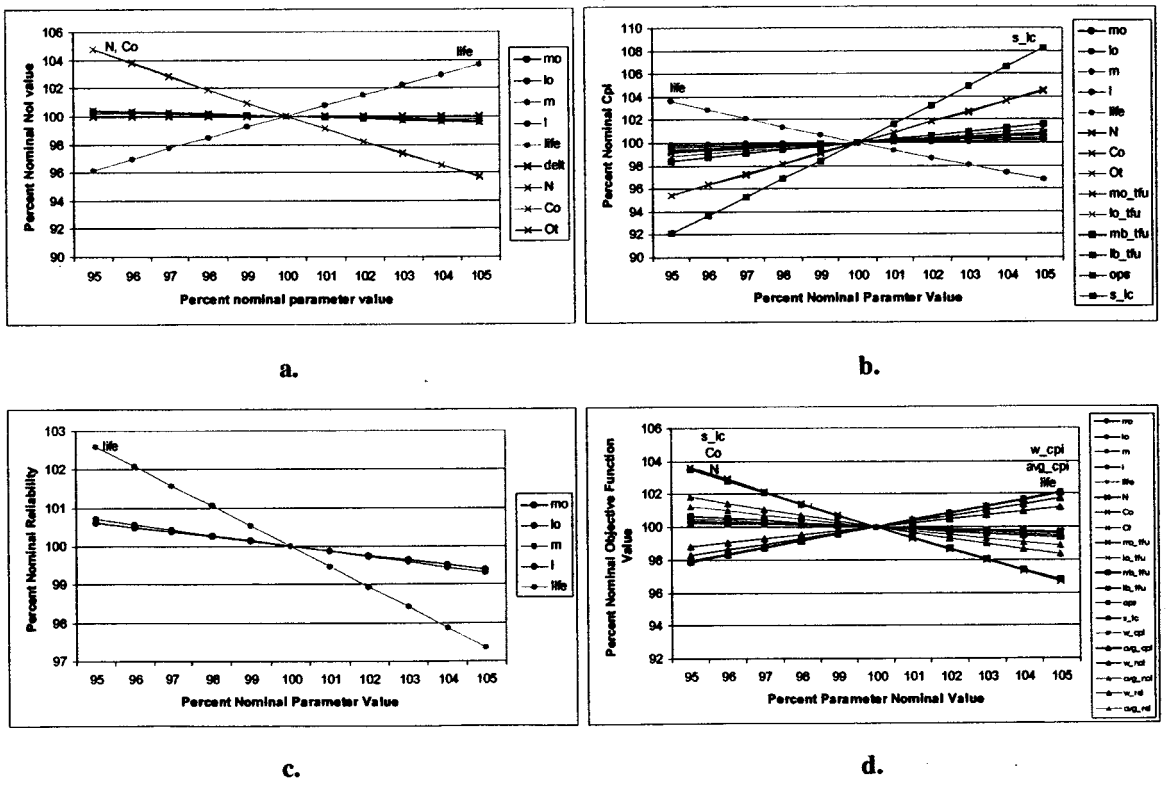


Figure 5.8 Sensitivity analysis results for zero dual functioning, two combining, and five collecting spacecraft architecture with no money spent on improving reliabilities.

The parameters with the largest effect on each metric are labeled in Figure 5.8. The mission design lifetime, number of pairs of UV points needed per image, and integration time

per pair of UV points have the largest effect on the number of images. The mission design lifetime and learning curve slope have the largest effect on the cost per image. The mission design lifetime has the largest effect on the reliability. Finally, all four of these parameters (mission design lifetime, number of pairs of UV points, integration time per pair of UV points, and learning curve slope) plus the weighting factor of the cost per image in the objective function definition and the average value used to scale the cost per image have the largest effect on the objective function value. It makes sense that the weight attached to the cost per image and average cost per image have a larger effect on the objective function than the weight on and average of the number of images and the reliability, since the cost per image has a larger initial weighting (0.4 versus 0.3 for the other two metrics). The mission design lifetime has one of the largest effects of all the parameters on all four of the metrics shown in Figure 5.8. Therefore, a more detailed sensitivity analysis was done focusing only on the mission design lifetime. The results for the sensitivity of the number of images to the mission design lifetime are shown in Figure 5.9.

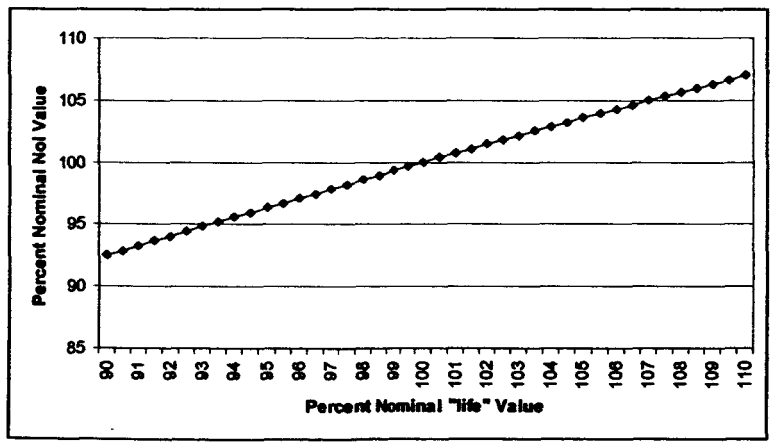


Figure 5.9 Detailed sensitivity analysis for mission lifetime versus expected total number of images (NoI)

If there were no failures in the system, it would be expected that the number of images would scale directly with the lifetime of the mission, such that a mission with a 10% longer lifetime would produce 10% more images. Once failures are taken into account

however, this relationship is not nearly as simple. If failures have occurred, the system will not be as productive later in life as it was in the beginning of the mission. Therefore a 10% increase in the mission lifetime should result in a less than 10% increase in the number of images produced. Similarly, a 10% decrease in the mission lifetime should result in a less than 10% decrease in the total number of images produced. In addition, a larger percentage decrease in the number of images is expected when a mission lifetime is shortened then a similar increase in the number of images if a mission lifetime is increased by the same percent. This is due to the failures expected to occur, making the system less productive, in the time from the shortened lifetime to the nominal lifetime and again from the nominal lifetime to the extended lifetime. Figure 5.8 shows a 7% increase in the number of images if the mission lifetime is extended 10%, and a 7.5% decrease in the number of images if the mission lifetime is decreased by 10%, matching with the previous predictions.

One of the major questions that the sensitivity analysis tool is trying to answer is whether or not changes in the user inputted parameter values would cause a change in which architecture is reported as "optimal". This can be answered by comparing the sensitivity of multiple architectures to the same changes in the design parameters. The architecture reported by the genetic algorithm with an objective function value nearest to the objective function value of the "optimal" solution, with the number of at least one type of spacecraft different from the "optimal" solution, has one dual functioning spacecraft, one combining spacecraft, and four collecting spacecraft. A sensitivity analysis of this architecture, with no money spent on improvements to component reliabilities, was carried out to compare with the results of the sensitivity analysis of the "optimal" solution. The full results of this comparison can be seen in Appendix D, and a sampling of the results can be seen in Figure 5.10.

In Figure 5.10a and Figure 5.10b, the sensitivity lines for the two different architectures are parallel to one another. Therefore, if one of parameters the user reported was not correct, or changed for one reason or another, while both architecture's objective functions

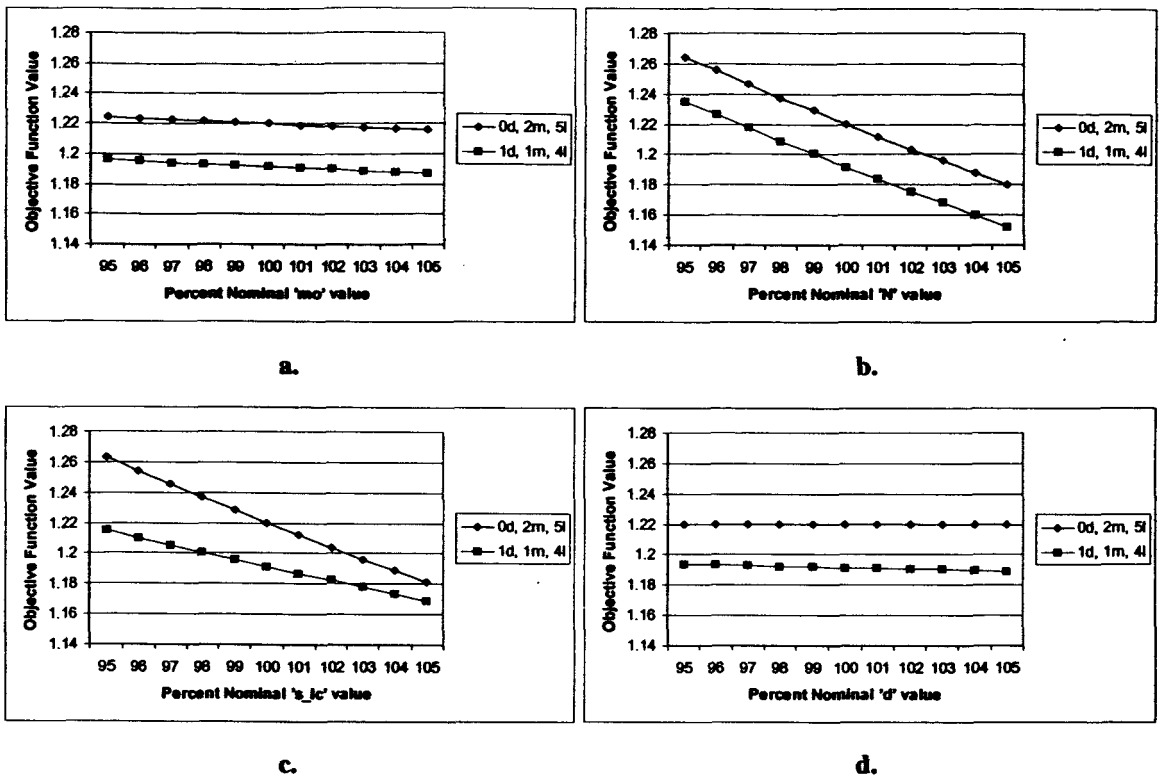


Figure 5.10 Sensitivity analysis for both 0 dual functioning, 2 combining, and 5 collecting spacecraft and 1 dual functioning, 1 combining, and 4 collecting spacecraft architectures. Sensitivity of objective function value to a) combining optics failure rate, b) number of pairs of UV points needed per image, c) learning curve slope, and d) dual bus failure rate.

would change, they would change by the same amount. This implies that the architecture which was reported as "best" or "optimal," with the highest objective function value, would be reported as the "optimal" solution no matter what the parameter value was set at. While the sensitivity lines of the objective function to the majority of parameters are parallel in this fashion, there are parameters in which this is not the case. If the sensitivity lines of the objective function to one of the parameters are not parallel, such as is the case in Figure 5.10c and Figure 5.10d, the "optimal" architecture could change, depending on the parameter value. Note that the y-axes in Figure 5.10 are objective function value, and not percent change in objective function value. This allows the user to find the parameter value at which the architecture that began at a lower objective function value would become the "better" architecture, with a higher objective function value than the other architecture. The sensitivity lines of the objective function value to the dual functioning

spacecraft bus failure rate are not parallel, as shown in Figure 5.10d. These lines however will not cross for significant change in the parameter value, implying that the dual functioning spacecraft failure rate would need to be very far off from the nominal value to change the "optimal" solution, and therefore the "optimal" solution is fairly robust to this parameter. The sensitivity lines of the objective function value to the learning curve slope are also not parallel, as shown in Figure 5.10c, and are near intersection by 5% above the parameter value. This parameter would therefore need to be examined in greater detail to ensure that the reported "nominal" value was within approximately 5% of the actual value before running any optimization algorithm or comparing any architectures using this model. The default learning curve slope of 95% was suggested by Wertz and Larson for use with systems in which less than 10 units will be built [Wertz and Larson, 1999].

Sensitivity analyses were also carried out on the two architectures discussed above with the money spent to improve reliabilities of the components included in the architecture definition. The two architectures compared were the architectures reported by the genetic algorithm optimization scheme as the "optimal" architecture (zero dual functioning spacecraft, two combining spacecraft, five collecting spacecraft, $12M spent to improve the reliability of the combining optics, $9M spent to improve the reliability of the collecting optics, and $25M spent to improve the reliability of the bus) and the architecture with the second highest objective function value while still differing from the "optimal" solution by the number of at least one type of spacecraft (one dual functioning spacecraft, one combining spacecraft, four collecting spacecraft, $12M spent to improve the reliability of the combining optics, $13M spent to improve the reliability of the collecting optics, and $30M spent to improve the reliability of the bus). These two architectures were only analyzed for the sensitivity of the life cycle metrics to 10 of the 23 parameters. The ten parameters tested were the dual functioning spacecraft bus failure rate (*d*), the mission design lifetime (*life*), the scale factor used to map the money spent to improve the reliability of a component to the actual reliability increase (*S*), the learning curve slope (*s_lc*), and the weighting and average values for each of the three life cycle metrics used in the objec-

tive function (*w_cpi, avg_cpi, w_noi, avg_noi, w_rel, avg_rel).* The full results of these tests can be seen in Appendix D, and a sampling of the results can be seen in Figure 5.11.

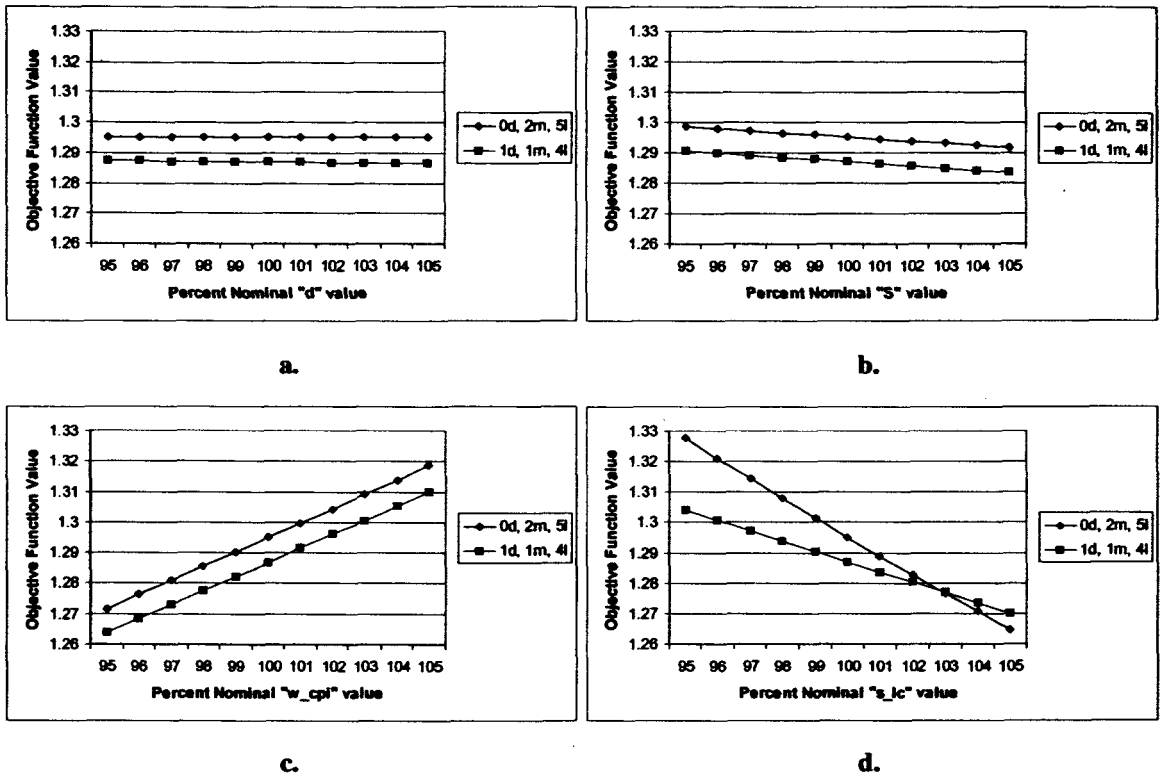


Figure 5.11 Sensitivity analysis for both 0 dual functioning, 2 combining, and 5 collecting spacecraft and 1 dual functioning, 1 combining, and 4 collecting spacecraft architectures when money spent to improve reliabilities of components is included in the design vector. Sensitivity of objective function value to a) dual functioning spacecraft bus failure rate, b) scale factor used to map money spent to improvement in reliability, c) weight of cost per image in objective function value, and d) learning curve slope.

While the majority of the sensitivity lines of the objective functions to the different parameters were closer together in this case study than in the previous case study (with no money spent to improve the reliability of components), the majority of lines in this case study are once again parallel, or nearly parallel. This is true for both the scale factor used to map the amount of money spent to improve reliabilities to the actual increase in reliability, as shown in Figure 5.11b, and the weightings and average value for each of the life cycle metrics. The sensitivity lines of the objective function to the weighting placed on the cost per image in the objective function are shown in Figure 5.11c, as an example of

this last category of parameters. The results for the other metric weightings and average values can be seen in Appendix D. The fact that the sensitivity lines for these parameters are parallel, or nearly parallel, for these two architectures once again shows that the "best" or "optimal" architecture will not change if the nominal value of these parameters change. The sensitivity lines for the objective functions to the dual functioning spacecraft bus failure rate are shown in Figure 5.11a, and are once again not parallel. It is clear however, in a similar fashion as was shown in the previous case study, that the two lines will not cross for any reasonable change in the parameter value, implying that the dual functioning spacecraft failure rate would need to be very far off from the nominal value to change the "optimal" solution, and therefore the "optimal" solution is fairly robust to this parameter. The sensitivity lines of the objective function value to the learning curve slope are shown in Figure 5.11d, and intersect at 103% of the nominal parameter value. This corresponds to a learning curve slope of 97.85%. This once again reinforces the need for this parameter to be examined in close detail before running any optimization algorithm or comparing any architectures using this model.

The sensitivity analysis tool developed can be tuned by the user to a wide range of sensitivity analysis options. These sensitivities can then be used to examine the robustness of designs and the effect of changing a user defined parameter on the outcome of the model and analysis tools discussed earlier. This type of sensitivity analysis can be especially important in the early stages of conceptual design, when most of the tools developed in this research are meant to be used, since the majority of the user defined inputs are very uncertain and may change drastically by the time the design is completed. The sensitivity analysis tools shown here will also help the designer to better learn which parameters to focus efforts on during the conceptual design phase, either to improve the parameter values or to reduce the uncertainty of the parameter, since they will effect the design outcomes more than other parameters. This was shown above with the example of the learning curve slope.

5.3 Chapter Summary

This chapter has presented several tools to aide the user in effectively searching the entire design space to find the "optimal" architecture, in terms of the number of each type of spacecraft and the money spent to improve individual component reliabilities, to achieve the highest total performance possible for a given budget. Two different heuristic algorithms, simulated annealing and genetic algorithms, were presented along with examples of their use. Genetic algorithms proved to be a very powerful design tool for the early stages in the design process, as this optimization scheme was able to return not only the "optimal" solution, but also several other unique and different architecture solutions which provide similar performance. Finally, a sensitivity tool was presented along with several case studies showing the various uses and insights that can be gained from this type of analysis.

Chapter 6

CONCLUSION

As technological systems grow in capability, they also grow in complexity. The number of components and interactions inherent in most of these systems today makes the design process quite challenging. With these complex systems, it is no longer possible for a designer to use engineering judgement to identify the components that have the largest impact on the life cycle metrics of the system as a whole, such as reliability, productivity, cost, and cost-effectiveness. The decisions made early in the design process however, including decisions of which components to focus efforts on, affect almost all aspects of the system further on in the design process. Therefore, it is imperative when dealing with complex systems to make the high-level architecture decisions early in the design process that will most cost effectively improve the life cycle metrics of the system as a whole. One way of accomplishing this is to build quantitative models and analysis tools which can be used to aid the designer in making these conceptual design decisions. Additionally, when missions are considered high risk, or are dealing with new and uncertain technologies, it is also important to ensure that these models capture the system behavior in the event of failures.

Once the key components that affect system life cycle metrics have been identified, two main approaches to improving the system using these components exist: adding redundancy or improving the reliability of the component. In actuality, the most effective approach to almost any system will be some combination of these two approaches in vary-

ing orders of magnitude for each component. The exact combination and magnitudes for each component is not clear to the designer without quantitative models and analysis tools. Therefore this research attempts to answer the question of how to divide funds, between adding redundancy and improving the baseline reliability of components, to most cost effectively improve the life cycle metrics of a system.

The first step in the process of answering the question posed above was to develop the models needed to analyze each architecture. To accomplish this, reliability, productivity, and cost models were developed for the application used throughout this research - separated spacecraft interferometers. These models all rely on the state-transition matrix - the matrix which defines the probability that the system is in each state throughout the life of the mission. A tool was developed, and verified, to automatically generate this state-transition matrix, given the rules of system failure. This tool, along with several of the methods used to calculate the total expected productivity (number of images), reliability, and cost-effectiveness (cost per image), could be used to analyze almost any space system by modifying the rules of system failure, the definitions of the states of the system, and the models for the productivity of the system in each state. The models and tools developed were tested using a variety of case-studies.

In the next step, a new metric was introduced to compare the total performance, by coupling all of the life cycle metrics described above, of different user-defined architectures. This metric was tested, and results were shown using two different case studies, both with and without money spent to improve the reliability of components. It is worth noting that the architecture with the best total performance is not necessarily the architecture with the best performance in any of the three individual life cycle metrics. Through these case studies, trends were identified which give general rules of thumb on how to divide a given amount of money among different components to improve individual reliabilities in order to improve the overall system reliability as much as possible.

The final step was to develop tools to aide the user in effectively searching the entire design space to find the "optimal" architecture in terms of the number of each type of spacecraft and the money spent to improve individual component reliabilities to achieve the highest total performance possible for a given budget. Two different heuristic algorithms, simulated annealing and genetic algorithms, were presented along with examples of their use. Genetic algorithms proved to be a very powerful design tool for the early stages in the conceptual design process, as this optimization scheme was able to return not only the "optimal" solution, but also several other unique and different architecture solutions which provide similar levels of performance. Finally, a sensitivity tool was presented along with several case studies showing the various uses and insights that can be gained from this type analysis.

6.1 Contributions

The objective of this research was to develop models and tools to help answer high-level architecture conceptual design questions in order to design a system with the highest life cycle throughput for a given cost. The specific contributions of the research are divided into two categories - tool development and case study results.

Tool Development

- State-transition matrix - An automatic state-transition matrix generation Matlab tool was developed. This tool can be used to generate a state-transition matrix for any system given the rules of system failure.
- Modeling - Models for productivity, reliability, and cost analysis for separated spacecraft interferometry systems were developed. The method of calculating reliability, as well as the method of integrating productivity and cost through the mission lifetime to determine a total expected productivity and cost, is applicable to all space systems. Models for the productivity and cost of the system in each state are specific to SSI systems.
- Comparison - A tool was developed to compare multiple user defined architectures in terms of total performance. These architectures can be defined in terms of either just the number of each type of spacecraft, or the number of each type of spacecraft and the total system budget with any money not

spent on buying and operating spacecraft spent on improving the reliability of individual components.

- Optimization - Tools were developed to implement two heuristic optimization techniques to search the entire design space for the "optimal" SSI architecture, in terms of total performance for a given budget, where architectures were defined by both the number of each type of spacecraft and the money spent to improve the reliability of each component. Simulated annealing and genetic algorithm tools were implemented.
- Sensitivity analysis - Sensitivity analysis tools were developed to find the sensitivity of any architecture, defined either in terms of simply the number of each type of spacecraft or the number of each type of spacecraft and the money spent to improve component reliabilities, to any of the user-defined constant parameters used in the models and tools developed above.
- Toolbox - All the models and tools described above were integrated into a single Matlab toolbox. Table 6.1 lists the most important files found in the toolbox, with a brief description of each. For a complete listing of all files in the toolbox, along with a description of all the user defined inputs, or parameters, please see Appendix E.

SSI Case Study Results

- When dividing money to be spent on improving individual component reliabilities for a given architecture (defined by the number of each type of spacecraft), more money should be spent improving the reliability of the collecting optics than improving the reliability of the combining optics only when there are two or more additional spacecraft capable of collecting light than there are capable of combining light.
- When dividing money to be spent on improving individual component reliabilities for a given architecture (defined by the number of each type of spacecraft), the rules for when to spend more money improving the reliability of the bus than the reliability of either set of optics change with the total number of spacecraft and the total system budget. Rules of thumb can be developed for each of these cases however.
- For a total system budget of $360M, the "optimal" architecture, in terms of total performance, has zero dual functioning spacecraft, two combining spacecraft, and five collecting spacecraft, with $12M spent to improve the reliability of the combining optics, $9M spent to improve the reliability of the collecting optics, and $25M spent to improve the reliability of the bus. An additional seven architectures have been identified, unique in the number of each type of spacecraft, which have a total objective function value within 97.5% of the "optimal" objective function value. These eight architectures

TABLE 6.1 File descriptions from Reliability and Productivity Matlab toolbox.

Filename	Description
Modeling Tools	
inputs.m	Contains all user defined inputs, or parameters. Called in other functions to set the values of these parameters.
DV_to_J.m	Finds the objective function, number of images, reliability, and cost per image of a given architecture.
state.m	Recursive function which generates the state-transition matrix for a given system.
Comparison Tools	
arch_ comparison.m	Compares user given architectures in terms of total performance. Spends no money on improving component reliabilities.
arch_ comparison_w_impr.m	Compares user given architectures in terms of total performance. Includes optimal division of money to improve component reliabilities.
Optimization Algorithms	
sim_annealing.m	Finds the "optimal" architecture, in terms of performance for a given budget, defined by the number of each type of spacecraft and the money spent to improve the reliability of each component. Uses simulated annealing optimization algorithm.
J_GA.m	Calls genetic algorithm program to find the "optimal" architecture, in terms of performance for a given budget, defined by the number of each type of spacecraft and the money spent to improve the reliability of each component.
Sensitivity Analysis	
sensitivity.m	Finds the sensitivity of a given architecture to user defined parameters (does not include money to improve component reliabilities).
sensitivity_full_x.m	Finds the sensitivity of a given architecture to user defined parameters (does include money to improve component reliabilities).

can now be compared in terms of less quantifiable metrics, such as risk, political effect, or heritage.

6.2 Future Work

The research presented here is the first step to solving a complex space system conceptual design problem. While several of the tools developed here are the building blocks required to continue to examine the questions posed, there is still a lot of work that can be done in this area of research. A list of some of the possibilities for future work in this problem is shown below.

- Improved model fidelity - The fidelity of the models developed in the research presented here could be improved. The accuracy of the results from any of the analysis tools will improve greatly with improved fidelity of the models used. Modeling of lower level components, addition of repair, improved cost models, and improved estimates of user defined parameters will lead to higher fidelity models, and therefore more accurate results.
- Uncertainty analysis - All of the analysis presented in this research depends on several user defined parameters. Several of these parameters are highly uncertain, such as the failure rates of the components. An analytic sensitivity analysis to such parameters could provide more accurate results than those reported in this research. In addition, propagating the uncertainty inherent in these parameters through the analyses could give the designer a better feel for how this uncertainty will affect the design.
- Robust design - In addition to uncertainty propagation, the uncertainty in the user defined parameters also implies the need of the designer to take robust design issues into account when making design decisions. Several robust design tools could be incorporated into this research, including Monte Carlo analyses, reliability analyses, and Taguchi methods.
- Multiobjective optimization - Multiobjective optimization techniques, such as solving for the Pareto front and ranking solutions in terms of domination, could help to take away any dependence of the solutions returned from the optimization programs to the weighting factors and average values of the different metrics used in the current objective function.
- Improved optimization algorithms - Hybrid optimization algorithms, such as using a heuristic technique followed by a gradient based technique, can help to reduce the possibility of a solution being returned which is not at least a local optimum.
- Applications - The optimization tools presented here could be used on a wide array of other total system budgets. This could allow designers to begin to see trends in the solutions and to develop generic rules of thumb, or heuristics, for how to divide money among redundancy and improved reli-

ability to achieve the best performance possible. In addition, more experiments involving the optimization of just the division of money to improve reliabilities among components would lead to stronger trends, and therefore improved rules of thumb, for this category of problem as well.

REFERENCES

[Babcock, 1986] Babcock, P.S., *An Introduction to Reliability Modeling of Fault-Tolerant Systems*, The Charles Stark Draper Laboratory, Inc., Technical Report, September 1986.

[Belanger, 1995] Belanger, P., *Control Engineering: A Modern Approach*, Saunders College Publishing, Philadelphia, PA, 1995.

[deWeck, 2002] deWeck, O., *Genetic Algorithms Basic Introduction*, MIT 16.899 Multidisciplinary System Design Optimization Lecture Notes, 2002.

[GSFC, MAXIM, 2001] Goddard Space Flight Center, "Micro-Arcsecond X-ray Imaging Mission", Goddard Space Flight Center [Online], available at http://maxim.gsfc.nasa.gov/maxim.html, July 2001.

[GSFC, SI, 2001] Goddard Space Flight Center, "The Stellar Imager (SI) Homepage", Goddard Space Flight Center [Online], available at http://hires.gsfc.nasa.gov/~si/, July 2001.

[GSFC, SPECS, 2001] Goddard Space Flight Center, "Submillimeter Probe of the Evolution of Cosmic Structure", Goddard Space Flight Center [Online], available at http://space.gsfc.nasa.gov/astro/specs/, July 2001.

[Holland, 1975] Holland, J., *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.

[Houck, Joines, and Kay, 1995] Houck, C., Joines, J., Kay, M., "A Genetic Algorithm for Function Optimization: A Matlab Implementation", available online at http://www.ie.ncsu.edu/gaot/, 1995.

[INCOSE, 1998] INCOSE, *Systems Engineering Handbook*, San Fransisco Bay Area Chapter International Council on Systems Engineering, Technical Report, January 1998.

[Jilla, 2000] Jilla, C., *Reliability Analysis for TPF*, Massachusetts Institute of Technology Space Systems Laboratory, Memorandom, June 2000.

[Jilla, 2002] Jilla, C., *Simulated Annealing*, MIT 16.899 Multidisciplinary System Design Optimization Lecture Notes, 2002.

[JPL, LF, 2001] Jet Propulsion Laboratory, "Life Finder (LF)" Jet Propulsion Laboratory [Online], available at http://origins.jpl.nasa.gov/missions/lf.html, July 2001.

[JPL, LISA, 2001] Jet Propulsion Laboratory, "Laser Interferometer Space Antenna", Jet Propulsion Laboratory [Online], available at http://lisa.jpl.nasa.gov/, July 2001.

[JPL, TPF, 2001] Jet Propulsion Laboratory, "Terrestrial Planet Finder (TPF)", Jet Propulsion Laboratory [Online], available at http://origins.jpl.nasa.gov/missions/tpf.html, July 2001.

[Kirkpatrick, Gelatt, and Vecchi, 1983] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., "Optimization by Simulated Annealing", *Science,* Volume 220, Number 4598, 13 May 1983, pp. 671-680.

[Lay, 2001] Lay, O. Interviews by author. June-August 2001.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, M., Rosenbluth, A., Teller, A. and Teller E., "Equation of State Calculations by Fast Computing Machines", *J. Chem. Phys.*, Volume 21, Number 6, 1953, pp. 1087-1092.

[Miller, 2001] Miller, D. Interview by author. July 2001.

[Selby, 1971] Selby, S.M., *Standard Mathematical Tables - Nineteenth Edition*, The Chemical Rubber Co., Clevland, OH, 1971.

[Shaw, Miller, and Hastings, 2000] Shaw, G.B., Miller, D.W. and Hastings, D.E., "Generalized Characteristics of Satellite Systems", *Journal of Spacecraft and Rockets,* Vol. 37, No. 6, 2000, pp.801-811.

[Strang, 1986] Strang, G., *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.

[Wertz and Larson, 1999] Wertz, J., and Larson, W., *Space Mission Analysis and Design*, Kluwer Academic Publishers, Boston, MA, 1999.

Appendix A

RELIABILITY AND PRODUCTIVITY TOOLBOX SOURCE CODE

A Matlab toolbox containing all the tools and models previously described has been assembled. The source code for the major files contained in the toolbox is shown below. Please see Appendix E for a listing and description of all files and the user-defined inputs in the Reliability and Productivity toolbox.

A.1 "state.m"

```
%This function automatically generates an A matrix for
%any architecture (number of duals, combiners, and
%collectors). The logic and process used in this code
%could easily be modified to generate an automatic
%A matrix for any system. It is a recursive function.
%This function finds the A matrix for the system assuming
%a failure rate is known for collecting optics, combining
%optics, collector bus, combiner bus, and dual functioning
%bus. The collector and combiner are assumed to fail when
%either their optics or bus fail. The dual functioning
%spacecraft can fail in three modes. If the dual functioning
%spacecraft collector optics fail, the spacecraft becomes a
%combiner. If the dual functioning spacecraft combiner
%optics fail, the spacecraft becomes a collector. If the
%dual functioning spacecraft bus fails, the whole spacecraft
%fails.

%INPUTS
%a: A matrix - originally this consists of just the first
%entry (row1, col1), however when it is recursively called
%this allows the A matrix to updated.

%d,m,l: Number of dual functioning, combining, and
%collecting spacecraft respectively
```

```
%last_state: Number of previous state for keeping track
%of where each state came from

%dml: The dml matrix has a row for each state and the
%number of duals, combiners, and collectors in each
%column, respectively

%f_mo,f_lo,f_d,f_m,f_l: Failure rates for combining and
%collecting optics, and dual functioning,
%combining, and collecting buses respectively

function[a,dml]=
state(a,d,m,l,last_state,dml,f_mo,f_lo,f_d,f_m,f_l)

%Set initial conditions
i = 1;
state_num = 0;

%Define a state vector, dml_new
dml_new = [d m l];
state1 = 0;
%if dml(1,:) = [0 0 0] then this must be the first time
%through state.m
if dml(1,:) == [0 0 0]
number_of_states = 0; %Therefore, there are 0 present states
state1 = 1; %This is the first state.  If state1 == 1
   %then don't change the current entry in the a matrix
  %but continue with the recursive process.
else  %Otherwise find the number of present states
[number_of_states,junk] = size(dml);
end

%Check to see if the current state has already been
%identified and given a state number
while i <= number_of_states
if dml(i,1) == dml_new(1)
if dml(i,2) == dml_new(2)
if dml(i,3) == dml_new(3)
state_num = i;
end
end
end
i = i+1;
end

%If the current state hasn't been identified before,
%give it a state number and add it to the dml matrix
if state_num == 0
state_num = number_of_states + 1;
dml(state_num,:) = dml_new;
end
```

```
%Check to see what has changed from the last state
%i.e. if change == 1.3, then a dual bus has failed since the
%last state.
if dml_new(1) ~= dml(last_state,1)
   if dml_new(2) ~= dml(last_state,2)
      change = 1.1;   %If change = 1.1 then dual collecting
%optics have failed
   elseif dml_new(3)~= dml(last_state,3)
      change = 1.2;   %If change = 1.2 then dual combining
%optics have failed
   else
      change = 1.3;   %If change = 1.3 then a dual bus has
%failed
   end
elseif dml_new(2) ~= dml(last_state,2)& dml_new(1) ==
dml(last_state,1)
   change = 2;   %If change = 2 then a combiner has failed
%(optics or bus)
elseif dml_new(3) ~= dml(last_state,3)& dml_new(1) ==
dml(last_state,1)
change = 3;   %If change = 3 then a collector has failed
%(optics or bus)
elseif state1 == 1
change = 0;   %If change = 0 then this is the first state
%and no change should be made
else
error('No change in state')
end

a(state_num,state_num)=(d*f_d+d*f_mo+d*f_lo+m*(f_m+f_mo)+l*(
f_l+f_lo));
if change == 1.1
   a(state_num, last_state)=(d+1)*f_lo;
elseif change == 1.2
   a(state_num, last_state) = (d+1)*f_mo;
elseif change == 1.3
a(state_num, last_state) = (d+1)*f_d;
elseif change == 2
a(state_num,last_state) = (m+1)*(f_m+f_mo);
elseif change == 3
a(state_num,last_state) = (l+1)*(f_l+f_lo);
end


d_new = d-1;
m_dnew = m+1;
m_new = m-1;
l_dnew = l+1;
l_new = l-1;
```

```
%Only continue down "branch" if all "rules" are acceptable
%(i.e. another failure will not lead to system failure) AND
%if this "branch" has not been recorded before (i.e. this is
%the first time this state has been seen)
if d+l > 2 & d+m > 1 & d+m+l> 3 & d >= 1 & state_num >
number_of_states
[a,dml]=
state(a,d_new,m,l,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end

if d+l > 2 & d+m >= 1 & d+m+l> 3 & d >= 1 & state_num >
number_of_states
[a,dml]=
state(a,d_new,m_dnew,l,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end

if d+l >= 2 & d+m > 1 & d+m+l> 3 & d >= 1 & state_num >
number_of_states
[a,dml]=
state(a,d_new,m,l_dnew,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end

if d+l >=2 & d+m > 1 & d+m+l> 3 & m >= 1 & state_num >
number_of_state
[a,dml]=
state(a,d,m_new,l,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end

if d+l > 2 & d+m >= 1 & d+m+l > 3 & l >= 1 & state_num >
number_of_states
[a,dml]=
state(a,d,m,l_new,state_num,dml,f_mo,f_lo,f_d,f_m,f_l);
end
```

A.2 "DV_to_J.m"

```
%This Matlab code takes in a design vector (DV)
%and produces the expected total number of
%images (NoI) the system will produce in the
%given lifetime, the reliability of the system,
%the cost per image, the state-transition matrix,
%and the objective function.  This is accomplished by
%calculating the probability of being in each
%state of the system throughout time and using
%a model to predict the producitivity and cost of each
%of these states.  The probability of being
%in each state is then multiplied by the
%productivity or cost of the state and all states
%are summed to get the total expected productivity
```

```
%for that time period.
%A state is defined as a set of each of the
%satellites in either a failed or unfailed
%mode.  The design vector is defined as
%the number of dual functioning spacecraft,
% the number of combining spacecraft, the
%number of collecting spacecraft, the money
%spent to improve the reliability of the
%combining optics in millions of dollars,
%the money spent to improve the reliability of the
%collecting optics in millions of dollars, and
%the money spent to improve the reliability of the
%bus in millions of dollars.  All system parameters
%are listed in the inputs.m file.
%-------------------------------------------------
%-------------------------------------------------

%INPUTS
%x = design vector (dual, com, col, Xmo, Xlo, Xb)
%where -
%dual = # of dual functioning spacecraft
%com = # of combining spacecraft
%col = # of collecting spacecraft
%Xmo = $M spent on improving reliability of combining optics
%Xlo = $M spent on improving reliability of collecting
%optics
%Xb = $M spent on improving reliability of bus

%OUTPUTS
%J = objective function
%NoI = Expected total number of images produced
%a = state-transition matrix
%Rel = Reliability of system
%Cpi = Cost per Image

%-------------------------------------------------

function [J, NoI, a, Rel, Cpi] = DV_to_J(x);

clear P P0 t col com dual A n n1 Nc Tc Ti Ci a I M NoI ci R

%Call out the elements of the design vector
dual = x(1);
com = x(2);
col = x(3);
Xmo = x(4);
Xlo = x(5);
Xb = x(6);

%-------------------------------------------------------------
-------------------------
```

```
%Call out the system parameters
inputs

%-------------------------------------------------------------
-------------------------
%Use local variables instead of global variables
mo = COMBINER_OPTICS_FAILURE_RATE;
%Failure rate of combiner optics in months^-1
lo = COLLECTOR_OPTICS_FAILURE_RATE;
%Failure rate of collector optics in months^-1
d = DUAL_BUS_FAILURE_RATE;
%Failure rate of dual functioning bus in months^-1
mb = COMBINER_BUS_FAILURE_RATE;
%Failure rate of combiner bus in months^-1
lb = COMBINER_BUS_FAILURE_RATE;
%Failure rate of combiner bus in months^-1
life = MISSION_DESIGN_LIFETIME;
%Mission design life in months
S = SCALE_FACTOR_FOR_INCREASE;
%Scale factor for increased reliability
com_req = REQUIRED_COMBINING_SPC;
%Number of required spacecraft acting as combiners
col_req = REQUIRED_COLLECTING_SPC;
%Number of required spacecraft acting as collectors
total_req = REQUIRED_TOTAL_SPC;
%Number of total spacecraft required
N = NUMBER_OF_DIFFERENCES;
%Number of independant differences (number of pixels/2)
Co = TIME_PER_CONFIGURATION;
%Time in months needed in each configuration
Ot = OVERHEAD_TIME_PER_IMAGE;
%Overhead time per image in months
B_tot = TOTAL_SYSTEM_BUDGET;
%Total system budget(spacecraft + improvements)
sf_cpi = WEIGHTING_FOR_CPI*AVERAGE_VALUE_FOR_CPI;
%Scale factor for weighting CPI
sf_noi = WEIGHTING_FOR_NOI/AVERAGE_VALUE_FOR_NOI;
%Scale factor for weighting NoI
sf_rel = WEIGHTING_FOR_REL/AVERAGE_VALUE_FOR_REL;
%Scale factor for weighting Reliability

%Set all metrics to worse possible if architecture does not
%function in initial state
if com + dual < com_req | col + dual < col_req | com + col +
dual < total_req
    Rel = 0;
    NoI = 0;
    Cost = 100000000;
    a=0;
else
```

```
%-------------------------------------------------------------
----------------------
%Calculate new reliabilities after money is spent to improve

%Change failure rates into reliabilities
R_mo = exp(-mo*life);
R_lo = exp(-lo*life);
R_d = exp(-d*life);
R_mb = exp(-mb*life);
R_lb = exp(-lb*life);

%Increase reliabilities given money spent to improve
R_mo_f = R_mo + (1-R_mo)*(1-exp(-Xmo/S));
R_lo_f = R_lo + (1-R_lo)*(1-exp(-Xlo/S));
R_mb_f = R_mb + (1-R_mb)*(1-exp(-Xb/S));
R_lb_f = R_lb + (1-R_lb)*(1-exp(-Xb/S));
R_d_f = R_d + (1-R_d)*(1-exp(-Xb/S));

%Change reliabilities back into failure rates
mo_f = -(log(R_mo_f))/life;
mb_f = -(log(R_mb_f))/life;
lo_f = -(log(R_lo_f))/life;
lb_f = -(log(R_lb_f))/life;
d_f = -(log(R_d_f))/life;

%-------------------------------------------------------------
------
%Generate the state-transition matrix

%The dml matrix is a matrix with a row for each state.  The
%columns are the number of duals, combiners, and collectors
%respectively in each state.  This matrix keeps tracks of the
%states.
dml(1,:) = [0 0 0];
%last_state is the number (or row number of the dml matrix)
%of the previous state
last_state = 1;
%The very first entry (1,1) of every a matrix is simply the
%failure rate of each component added together (times minus
%one)
a(1,1)   =   -(dual*d_f   +   dual*mo_f   +   dual*lo_f   +
com*(mo_f+mb_f) +col*(lo_f+lb_f));

count = 0;

%Function state returns the A and dml matrices for a given
% architecture.  It is a recursive function.  Please see
%state.m for further details.
```

```
[a,dml]                                                              =
state(a,dual,com,col,last_state,dml,mo_f,lo_f,d_f,mb_f,lb_f)
;

%-------------------------------------------------------------------
------
%Find the reliability and number of images

%Find the n vector.  This vector is the number of spacecraft
%acting as a collector in each state.  It is used to find the
%number of baselines for each state.
for i = 1:size(dml,1)
if dml(i,2) <= 0                    %If there are no combiners:
n(i) = (dml(i,1)-1) + dml(i,3);  %number of virtual
%collectors is the number of collectors (dml(i,3)) plus the
%number of duals (dml(i,1)) minus one to be used for the
%combiner
else              %If there are combiners:
n(i) = dml(i,1) + dml(i,3);    %number of virtual collectors
%is the number of collectors (dml(i,3)) plus the number of
%duals (dml(i,1))
end
end



Nb = (n.*(n-1))./2; %Number of baselines



Nc = ceil(N./Nb);       %Number of configurations needed

Ti = N*Co+Co*Nc+Ot; %Time needed in each step involves time
%per pair of pixels + time for # of configs
%+ overhead time
Ci = 1./Ti;  %Imaging rate in months^-1


num_states = length(a);
P0 = zeros(num_states,1);   %First creat initial conditions
P0(1) = 1;
p_final = expm(a*life)*P0;
%Integrate the number of images through the lifetime of the
%mission
NoI = Ci*inv(a)*(expm(a*life)-eye(size(a)))*P0;
%Reliability = probability that the system is in a working
%state at the end of the mission lifetime
Rel = sum(p_final);
```

```
%-------------------------------------------------------------
------
%Calculate cost
Cost = cost_model(dual,com,col,Xmo,Xlo,Xb,a,Nb);

end

%Cost per image is simply the total cost divided by the
%number of images
if NoI == 0
    Cpi = 10^9;
else
    Cpi = Cost/NoI;
end

%If the budget constraint is broken, the objective function
%= 0
if Cost > B_tot
    NoI=0;
    J=0;
%If budget constraint isn't broken - calculate objective
%function
else
    J = sf_noi*NoI+Rel*sf_rel+1/Cpi*sf_cpi;
end
```

A.3 "cost_model.m"

```
%This function calculates the total cost of
%a separated spacecraft interferometer system in $M.
%The function takes as inputs the number of
%each type of spacecraft (dual functioning,
%combining and collecting) and the amount of
%money spent to improve the reliability of the
%combining optics, collecting optics, and bus.
%All cost models are low fidelity and simply
%estimate the differences in costs between
%systems.  Operations costs are assumed to scale
%with the number of baselines.
%This is due to two factors: 1) more baselines
%lead to larger configurations needed
%to keep in formation, with requires more effort
%than smaller configurations, and 2) the larger
%the number of baselines, the faster
%the light can be collected, and therefore the sooner
%the cluster needs to be moved to another configuration.
%While this implies that clusters with more baselines
%can collect more images in the same time, it
%also implies that the operations cost for these clusters
```

```
%will be higher since there is more cluster movement.

function Cost = cost_model(dual, com, col, Xmo, Xlo,
Xb,a,Nb)

%Input definitions
%dual = number of dual functioning spacecraft
%com = number of combining spacecraft
%col = number of collecting spacecraft
%Xmo = money spent to improve combining optics reliability
%in $M
%Xlo = money spent to improve collectiong optics reliability
%in $M
%Xb = money spent to improve bus reliability in $M
%a = state transition matrix
%Nb = vector of number of baselines in each state

%Hard inputs
inputs

mo_tfu     =     COMBINER_OPTICS_THEORETICAL_FIRST_UNIT_COST;
%Cost of combiner optics theoretical first unit cost in $M
lo_tfu     =    COLLECTOR_OPTICS_THEORETICAL_FIRST_UNIT_COST;
%Cost of collector optics theoretical first unit cost in $M
mb_tfu      =      COMBINER_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of combiner bus theoretical first unit cost in $M
lb_tfu      =      COLLECTOR_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of collector bus theoretical first unit cost in $M
db_tfu = DUAL_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of dual functioning bus theoretical first unit cost in
%$M
ops = OPERATIONS_COST_PER_BASELINE;
%Operations cost per month per baseline ($M/month)
life = MISSION_DESIGN_LIFETIME;
%Mission design life in months
s_lc = LEARNING_CURVE_SLOPE;
%Learning curve slope percentage (see SMAD for recommended
%values)
%NOTE: Entered as 95 NOT 0.95


%Calculate manufactoring cost for each type of spc.

%Dual TFU cost = cost of dual bus + combining optics +
%collecting optics
%If more than one unit is made, use the learning curve
%percentage given to reduce the cost of future units
dual_cost = (db_tfu + mo_tfu + lo_tfu)*(dual^(1-(log(100/
s_lc)/log(2))));

%Com TFU cost = cost of com bus + combining optics
```

```
%If more than one unit is made, use the learning curve
%percentage given to reduce the cost of future units
com_cost    =    (mb_tfu    +    mo_tfu)*(com^(1-(log(100/s_lc)/
log(2))));

%Col TFU cost = cost of col bus + collecting optics
%If more than one unit is made, use the learning curve
%percentage given to reduce the cost of future units
col_cost    =    (lb_tfu    +    lo_tfu)*(col^(1-(log(100/s_lc)/
log(2))));



%Calculate operations cost
%Baseline operations costs are given in terms of cost per
%collecting spacecraft per month.  Therefore, each state of
%a system will have a different cost per month.  Therefore,
%the a-matrix is required to integrate this cost, taking
%into account the probability of being in each state
%throughout time, through the lifetime.
num_states = length(a);
P0 = zeros(num_states,1);   %First creat initial conditions
P0(1) = 1;
cost_per_state = ops*Nb;
ops_cost        =        cost_per_state*inv(a)*(expm(a*life)-
eye(size(a)))*P0;

%Total cost is manufactoring cost + operations cost + money
spent on improving reliabilities

Cost = dual_cost+com_cost+col_cost+ops_cost+Xmo+Xlo+Xb;
```

A.4 *"arch_comparison.m"*

```
%This file compares multiple user-defined architectures
%in terms of total performance.  Plots of the
%number of images, cost per image, reliablity, and
%total score for each architecture are returned.
%This code will only vary the number of
%satellites in the initial state and not the money
%spent to improve the reliability of any components.

%Matlab functions clock and etime are used.  See help
%files for both functions.  These files are only used
%to keep track of the time required to analyze an
%architecture
%-------------------------------------------------------
%-------------------------------------------------------
```

```
%INPUTS
%No function inputs
%Architecture to be evaluated and weightings
%for each life cycle metric are defined in inputs.m file

%OUTPUTS
%Plots of NoI, Reliability, CpI, and "score" for each
%architecture.  Each of these metrics can be reported
%for each architecture by calling the architecture and
%scores matrices. In addition, the architecture matrix
%contains the time required to analyze the system,
%the architecture itself, and the size of the A matrix.
%-------------------------------------------------

clear

inputs

archs = ARCHITECTURE_MATRIX;
SF_CPI = WEIGHTING_FOR_CPI;
%Scale factor for weighting CPI in objective function
SF_NoI = WEIGHTING_FOR_NOI;
%Scale factor for weighting NoI in objective function
SF_Rel = WEIGHTING_FOR_REL;
%Scale factor for weighting Rel in objective function

architecture = zeros(1,8);

for i=1:size(archs,1)

   clear NoI a count time_elapsed to num_calcs
    xo = [archs(i,1) archs(i,2) archs(i,3) 0 0 0];
   to = clock;
   [J,NoI,a,Rel,Cpi] = DV_to_J(xo);
   time_elapsed = etime(clock,to);
   num_calcs = full(sum(sum(spones(a))));

   %Save data for comparison after all architectures have
  %been evaluated
   architecture(i,:) =
    [xo(1) xo(2) xo(3) time_elapsed length(a) NoI Rel Cpi];
   Cost_per_Image(i) = Cpi;
   Number_of_Images(i) = NoI;
   Reliabilities(i) = Rel;

end

%Find best of each metric
Max_num_images = max(architecture(:,6));
Max_reliability = max(architecture(:,7));
Min_CPI = min(architecture(:,8));
```

```
%Calculate the scores for each architecture and output the
%"best" architecture
scores              =              SF_CPI.*(Min_CPI./architec-
ture(:,8))+SF_NoI.*(architecture(:,6)./
Max_num_images)+SF_Rel.*(architecture(:,7)./
Max_reliability);

max_score = max(scores)
score_index = find(scores == max_score);
best_duals = num2str(architecture(score_index,1));
best_coms = num2str(architecture(score_index,2));
best_cols = num2str(architecture(score_index,3));
disp(['The "best" architecture has ' best_duals ' dual func-
tioning spacecraft, ' best_coms ' combiners, and ' best_cols
' collectors.']);

figure      %Bar graph to compare cpi between archs.
bar(Cost_per_Image);
xlabel('Architecture')
ylabel('Cost per Image in $M')
title('Cost per image for various architectures')

figure      %Bar graph to compare # of images between archs.
bar(Number_of_Images);
xlabel('Architecture')
ylabel('Number of Images')
title('Number of images for various architectures')

figure      %Bar graph to compare # of images between archs.
bar(Reliabilities);
xlabel('Architecture')
ylabel('Reliability')
title('Reliability for various architectures')

figure      %Bar graph to compare # of images between archs.
bar(scores);
xlabel('Architecture')
ylabel('Relative score')
title('Relative scores for various architectures')
```

A.5 *"opim_reliability_w_test.m"*

```
%This function finds the optimal reliability possible given
%an architecture in terms of the number of each type of
%spacecraft and a total budget.  The function returns
%the amount of money to be spent on improving the
%reliability of each component (Xmo, Xlo, and Xb),
```

```
%the final failure rates of all components, and the
%total reliablity of the system.

function [Xmo, Xlo, Xb, mo_f, mb_f, lo_f, lb_f, d_f, R_f] =
optim_reliability_w_test(com, col, dual, budget_flag, perc,
B)

inputs
%User inputs
%com            %Number of combiners in architecture
%col            %Number of collectors in architecture
%dual           %Number of dual functioning spacecraft in
          %architecture
%budget_flag    %0 = Use percentage of cost of spacecraft to
              %find total amount to spend on improvements
              %1 = Use total budget minus cost of spacecraft
             %to find amount to spend on improvements
%perc           %Percentage to use if budget_flag = 0
         %(arbitrary if budget_flag = 1)
          %Enter as 20, not 0.20, etc.
%B              %Total amount of money to be spent on system
              %(including improvements and new spacecraft)
         %(arbitrary if budget_flag = 0)

%Hard inputs
mo = COMBINER_OPTICS_FAILURE_RATE;
%Failure rate of combiner optics in months^-1
lo = COLLECTOR_OPTICS_FAILURE_RATE;
%Failure rate of collector optics in months^-1
d = DUAL_BUS_FAILURE_RATE;
%Failure rate of dual functioning bus in months^-1
mb = COMBINER_BUS_FAILURE_RATE;
%Failure rate of combiner bus in months^-1
lb = COMBINER_BUS_FAILURE_RATE;
%Failure rate of combiner bus in months^-1
life = MISSION_DESIGN_LIFETIME;
%Mission design life in months
S = SCALE_FACTOR_FOR_INCREASE;
%Scale factor for increased reliability

%Outputs
%Xmo              %Money spent on improving combining optics
%Xlo             %Money spent on improving collecting optics
%Xb              %Money spent on improving bus
%mo_f       %Final combining optics failure rate in months^-1
%mb_b       %Final combining bus failure rate in months^-1
%lo_f      %Final collecting optics failure rate in months^-1
%lb_f     %Final collecting bus failure rate in months^-1
%d_f       %Final dual bus failure rate in months^-1
%R_f      %Final reliability of system
```

```
%The dml matrix is a matrix with a row for each state.  The
%columns are the number of duals, combiners, and collectors
%respectively in each state.  This matrix keeps tracks of the
%states.
dml(1,:) = [0 0 0];
%last_state is the number (or row number of the dml matrix)
%of the previous state
last_state = 1;
%The very first entry (1,1) of every a matrix is simply the
%failure rate of each component added together (times minus
%one)
a(1,1) = -(dual*d + dual*mo + dual*lo + com*(mo+mb)
+col*(lo+lb));


%Function state returns the A and dml matrices for a given
%architecture.  It is a recursive function.  Please see
%state.m for further details.
[a,dml]=state(a,dual,com,col,last_state,dml,mo,lo,d,mb,lb);

%-------------------------------------------------------------
%Find the reliability, number of images and cost

%Find the n vector.  This vector is the number of spacecraft
%acting as a collector in each state.  It is used to find the
%number of baselines for each state.
for i = 1:size(dml,1)
if dml(i,2) <= 0  %If there are no combiners:
n(i) = (dml(i,1)-1) + dml(i,3);  %number of virtual
             %collectors is the number of
                 %collectors (dml(i,3)) plus the number of
                 %duals (dml(i,1)) minus one to be used for
           %the combiner
else        %If there are combiners:
n(i) = dml(i,1) + dml(i,3);      %number of virtual
             %collectors is the number of
                %collectors (dml(i,3)) plus the number of
          %duals (dml(i,1))
end
end


Nb = (n.*(n-1))./2;  %Number of baselines


%Find budget to spend on improvements
if budget_flag == 0

    %Find total cost of system
```

```
    system_cost = cost_model(dual,com,col,0,0,0,a,Nb);

    Impr_B = (1+perc/100)*system_cost;
elseif budget_flag == 1
    Impr_B = B;
else
    error('Budget flag not set');
end

%Initialize optimal system cost
sys_cost_opt = 0;

%Set initial conditions for optimization problem
Xo = [0 0 0];   %Start with no money spent on improvements

while sys_cost_opt < 0.98*Impr_B
%Find optimal reliability
[best_Xmo,      best_Xlo,      best_Xb,      best_Rel]      =
sim_annealing_money_initial(dual,com,col,Impr_B,Xo);


Xmo = best_Xmo;
Xlo = best_Xlo;
Xb = best_Xb;
R_f = best_Rel;

clear dml Nb a last_state

%Change failure rates into reliabilities
R_mo = exp(-mo*life);
R_lo = exp(-lo*life);
R_d = exp(-d*life);
R_mb = exp(-mb*life);
R_lb = exp(-lb*life);

%Increase reliabilities given money spent to improve
R_mo_f = R_mo + (1-R_mo)*(1-exp(-Xmo/S));
R_lo_f = R_lo + (1-R_lo)*(1-exp(-Xlo/S));
R_mb_f = R_mb + (1-R_mb)*(1-exp(-Xb/S));
R_lb_f = R_lb + (1-R_lb)*(1-exp(-Xb/S));
R_d_f = R_d + (1-R_d)*(1-exp(-Xb/S));

%Change reliabilities back into failure rates
mo_f = -(log(R_mo_f))/life;
mb_f = -(log(R_mb_f))/life;
lo_f = -(log(R_lo_f))/life;
lb_f = -(log(R_lb_f))/life;
d_f = -(log(R_d_f))/life;
```

```
%The dml matrix is a matrix with a row for each state.  The
%columns are the number of duals, combiners, and collectors
%respectively in each state.  This matrix keeps tracks of the
%states.
dml(1,:) = [0 0 0];
%last_state is the number (or row number of the dml matrix)
%of the previous state
last_state = 1;
%The very first entry (1,1) of every a matrix is simply the
%failure rate of each component added together (times minus
%one)
a(1,1)    =    -(dual*d_f    +    dual*mo_f    +    dual*lo_f    +
com*(mo_f+mb_f) +col*(lo_f+lb_f));
%Function state returns the A and dml matrices for a given
%architecture.  It is a recursive function.  Please see
%state.m for further details.
[a,dml]=
state(a,dual,com,col,last_state,dml,mo_f,lo_f,d_f,mb_f,lb_f)
;

%-------------------------------------------------------------
%Find the reliability, number of images and cost

%Find the n vector.  This vector is the number of spacecraft
%acting as a collector in each state.  It is used to find the
%number of baselines for each state.
for i = 1:size(dml,1)
if dml(i,2) <= 0  %If there are no combiners:
n(i) = (dml(i,1)-1) + dml(i,3);  %number of virtual
             %collectors is the number of
                 %collectors (dml(i,3)) plus the number of
                 %duals (dml(i,1)) minus one to be used for
            %the combiner
else         %If there are combiners:
n(i) = dml(i,1) + dml(i,3);      %number of virtual
             %collectors is the number of
                %collectors (dml(i,3)) plus the number of
           %duals (dml(i,1))
end
end



Nb = (n.*(n-1))./2; %Number of baselines

sys_cost_opt = cost_model(dual,com,col,Xmo,Xlo,Xb,a,Nb);

Xo=[Xmo Xlo Xb];

end
```

A.6 "sim_annealing.m"

```
%This matlab function uses Simulated Annealing techniques
%to find the optimum architecture for a SSI system in order
%to maximize the objective function of the system.  An
%architecture
%is defined as the number of combiners, collectors, and
%dual functioning spacecraft and the money spent to
%improve the reliability of the combining and
%collecting optics and the bus.  The cooling schedule
%uses a guess at the change in J from neighbor to neighbor
%(delta_guess) to find the initial temperature such that
%the probability of jumping from a better state to a worse
%state
%is approximately 0.75.  The temperature is reduced by an
%equal
%amount either every iteration (if iter = step), or a user
%defined
%number of times (step).  Please note that iter needs to be
%divisible
%by step for this cooling schedule to work.

%The design vector is defined as:
%[dual com col Xmo Xlo Xb]
%where: dual = # of dual functioning spacecraft
%com = # of combining spacecraft
%col  = # of collecting spacecraft
%Xmo = money spent on improving combining optics ($M)
%Xlo = money spent on improving collecting optics ($M)
%Xb = money spent on improving bus ($M)

%INPUTS
%delta_guess = guess of how much J changes from one neighbor
        %to another
%iter = number of iterations total
%step = number of steps down in temperature
%NOTE: iter must be divisible by step!!!!

clear

inputs

B_tot = TOTAL_SYSTEM_BUDGET;
%Total system budget(spacecraft + improvements)
delta_guess = INITIAL_DELTA_GUESS_FOR_SA;
%Initial guess at difference between two neighboring design
%vectors' objective functions
iter = SA_NUMBER_OF_ITERATIONS;
%Total number of iterations through algorithm
step = SA_STEPS_DOWN_IN_TEMPERATURE;
%Total number of steps down in temperature
```

```
%Bounds and increments on design variables
dual_possible = SA_DUAL_FUNCTIONING_BOUNDS;
com_possible = SA_COMBINING_SPACECRAFT_BOUNDS;
col_possible = SA_COLLECTING_SPACECRAFT_BOUNDS;
Xmo_possible = SA_MONEY_ON_COMBINING_OPTICS_BOUNDS;
Xlo_possible = SA_MONEY_ON_COLLECTING_OPTICS_BOUNDS;
Xb_possible = SA_MONEY_ON_BUS_BOUNDS;


%First need to define the initial starting point
x = [2 2 2 0 0 0];
[J_original, NoIo, ao, Relo, Cpio]= DV_to_J(x);

%Next, need to define and initial Temperature and
%cooling schedule.  Initial temperature chosen to
%give an initial probability of going to a worse
%solution of approximately 0.75
Temp = ceil(-delta_guess/log(.75));

%Want the temperature to be very low at the end
%of all iterations.  Set the final temperature
%to be approximately 0.001

Temp_reduce = (0.001/Temp)^(1/step);
Runs_per_step = iter/step;
if Runs_per_step ~= round(Runs_per_step)
    error('Number of iterations does not go evenly into num-
ber of steps')
end

%Initialize the matrix data - matrix with all iterations
%data stored.  Also one with just the data
%used to move on to the next step stored.  Rows will be:
%[J temp dual com col Xmo Xlo Xb]
data = zeros(1,length(x)+2);
data_proceed = zeros(1,length(x)+2);

%Start the annealing loop
for k=1:1:step
for j = 1:1:Runs_per_step;
    data(size(data,1)+1,:) = [J_original  Temp x];

    %Use 2 DOF
    change_one = round(rand*length(x));
    change_two = round(rand*length(x));

    x_new = x;

    %Randomly change two elements of the design vector
    if change_one == 1
        new_index = round(rand*length(dual_possible));
```

```
    if new_index > 0
        x_new(1) = dual_possible(new_index);
    end
elseif change_one == 2
    new_index = round(rand*length(com_possible));
    if new_index > 0
        x_new(2) = com_possible(new_index);
    end
elseif change_one == 3
    new_index = round(rand*length(col_possible));
    if new_index > 0
        x_new(3) = col_possible(new_index);
    end
elseif change_one == 4
    new_index = round(rand*length(Xmo_possible));
    if new_index > 0
        x_new(4) = Xmo_possible(new_index);
    end
elseif change_one == 5
    new_index = round(rand*length(Xlo_possible));
    if new_index > 0
        x_new(5) = Xlo_possible(new_index);
    end
elseif change_one == 6
    new_index = round(rand*length(Xb_possible));
    if new_index > 0
        x_new(6) = Xb_possible(new_index);
    end
end

if change_two == 1
    new_index = round(rand*length(dual_possible));
    if new_index > 0
        x_new(1) = dual_possible(new_index);
    end
elseif change_two == 2
    new_index = round(rand*length(com_possible));
    if new_index > 0
        x_new(2) = com_possible(new_index);
    end
elseif change_two == 3
    new_index = round(rand*length(col_possible));
    if new_index > 0
        x_new(3) = col_possible(new_index);
    end
elseif change_two == 4
    new_index = round(rand*length(Xmo_possible));
    if new_index > 0
        x_new(4) = Xmo_possible(new_index);
    end
elseif change_two == 5
```

```
        new_index = round(rand*length(Xlo_possible));
        if new_index > 0
            x_new(5) = Xlo_possible(new_index);
        end
    elseif change_two == 6
        new_index = round(rand*length(Xb_possible));
        if new_index > 0
            x_new(6) = Xb_possible(new_index);
        end
    end

    %Calculate the new number of images
    %If total cost of system is greater than the budget
    %then the system is infeasible and there are no
    %images produced

    J_new = DV_to_J(x_new);


    %Save data for comparison after calculation
    data(size(data,1)+1,:) = [J_new  Temp x_new];

    %If the new design vector provides more images, go there
   if J_new >= J_original
        x = x_new;
        J_original = J_new;
    %Otherwise, only go to the new design vector with  a
    %probability of e^(-delta/Temp) (boltzman factor)
    else
        test = rand;
        delta = J_original-J_new;
        prob = exp(-delta/Temp);
        if test < prob
            x = x_new;
            J_original = J_new;
        end
    end

    %Also save a matrix of just the data used for the next
   %step
    data_proceed(size(data_proceed,1)+1,:) =
    [J_original Temp x];
    format compact
    %disp(j+(k-1)*Runs_per_step)
    fprintf(1,'%d %f',j+(k-1)*Runs_per_step);
end
    %Reduce the temperature by using the pre-determined
   %cooling schedule
    Temp = Temp*Temp_reduce;

end
```

```
%Find and output the best architecture found
best = data(find(max(data(:,1))==data(:,1)),:);

best_J = num2str(best(1,1));
best_duals = num2str(best(1,3));
best_coms = num2str(best(1,4));
best_cols = num2str(best(1,5));
best_Xm = num2str(best(1,6));
best_Xl = num2str(best(1,7));
best_Xb = num2str(best(1,8));

disp(['The final architecture has ' best_duals ' dual func-
tioning spacecraft, ' best_coms ' combiners, and ' best_cols
' collectors.']);
disp(['$' best_Xb 'M, $' best_Xm 'M, and $' best_Xl 'M dol-
lars should be spent to improve the bus, combiner optics, and
collector optics respectively.']);


x_opt = best(1,3:8);
J_opt = best(1,1);

%Finds the matrix good_obj_func, containing all
%architectures which have objective function values
%within 99% of the "optimal"
best_options = data(find(data(:,1)>=0.99*max(data(:,1))),:);
best_options = -1*sortrows(-1*best_options,1);
good_obj_func(1,:) = best_options(1,:);
opt = 1;
i = 2;
while isempty(opt)~=1
opt                                                    =
find(best_options(:,8)~=best_options(1,8)|best_options(:,7)~
=best_options(1,7)|...

best_options(:,6)~=best_options(1,6)|best_options(:,5)~=best
_options(1,5)|...

best_options(:,4)~=best_options(1,4)|best_options(:,3)~=best
_options(1,3));
if isempty(opt)~=1
good_obj_func(i,:) = best_options(opt(1),:);
best_options = best_options(opt(1):size(best_options,1),:);
i = i+1;
end
end

%Finds the matrix good_obj_func2, containing all
%architectures which have objective function values
%within 97.5% of the "optimal", but vary from this
```

```
%"optimal" solution by the number of at least one
%type of spacecraft.
good_options                                                    =
data(find(data(:,1)>=0.975*max(data(:,1))),:);
good_options = sortrows(good_options,1);
good_obj_func2(1,:) = good_options(1,:);
opt = 1;
i = 2;
while isempty(opt)~=1
opt = find(good_options(:,5)~=good_options(1,5)|...

good_options(:,4)~=good_options(1,4)|good_options(:,3)~=good
_options(1,3));
if isempty(opt)~=1
good_obj_func2(i,:) = good_options(opt(1),:);
good_options = good_options(opt(1):size(good_options,1),:);
i = i+1;
end
end
```

A.7 "J_GA.m"

```
%This matlab function uses Genetic Algorithm techniques
%to find the optimum architecture for a SSI system in order
%to maximize the objective function of the system.
%An architecture
%is defined as the number of combiners, collectors, and
%dual functioning spacecraft and the money spent to
%improve the reliability of the combining and
%collecting optics and the bus.  The Matlab genetic
%algorithm toolbox, GAOT, is used.

%The design vector is defined as:
%[dual com col Xmo Xlo Xb]
%where: dual = # of dual functioning spacecraft
%com = # of combining spacecraft
%col  = # of collecting spacecraft
%Xmo = money spent on improving combining optics ($M)
%Xlo = money spent on improving collecting optics ($M)
%Xb = money spent on improving bus ($M)

%INPUTS
%From inputs.m file
%NOTE - to change inputs, need to change inputs.m file
%in the GA folder of the Reliability and Productivity
%toolbox.
%num_of_gen = Number of generations
%num_in_pop = Population size
%xOver = Crossover rate
```

```
%mut = Mutation rate


clear
inputs

num_of_gen = NUMBER_OF_GENERATIONS;
%Number of generations to be evaluated
num_in_pop = POPULATION_SIZE;
%Population size per generation
xOver = CROSSOVER_RATE;
%Crossover (mating) rate
mut = MUTATION_RATE;
%Mutation rate

%First set the bounds of the problem
bounds=BOUNDS_FOR_GA;
%Bounds for design variables. Note variables listed in
%order:
%x = [dual com col Xmo Xlo Xb];

%Next need to initialize population.  This uses all default
%values and is only done to be able to get to future
%assignments when calling ga
initPop=initializega(num_in_pop,bounds,'DV_to_J_for_GA', [],
[1e-6 0]);

%Call to ga.  See ps file "A GA function optimization"
%pg. 9, section 4 for definitions
DATA_FOR_GA = zeros(1,7);
[x endPop bPop traceInfo] = ga(bounds,'DV_to_J_for_GA', [0],
initPop,...
      [1e-6  0  1], 'maxGenTerm', num_of_gen, 'tournSelect',
0.08, 'simpleXover',...
    xOver, 'binaryMutation', mut);
%Use binary with precision(epsilon) of 1 to instrict the
%integer constraints.

x
bPop
data = DATA_FOR_GA;

%Finds the matrix good_obj_func, containing all
%architectures which have objective function values
%within 99% of the "optimal"
best_options = data(find(data(:,1)>=0.99*max(data(:,1))),:);
best_options = -1*sortrows(-1*best_options,1);
good_obj_func(1,:) = best_options(1,:);
opt = 1;
i = 2;
while isempty(opt)~=1
```

```
opt                                                                    =
find(best_options(:,2)~=best_options(1,2)|best_options(:,7)~
=best_options(1,7)|...

best_options(:,6)~=best_options(1,6)|best_options(:,5)~=best
_options(1,5)|...

best_options(:,4)~=best_options(1,4)|best_options(:,3)~=best
_options(1,3));
if isempty(opt)~=1
good_obj_func(i,:) = best_options(opt(1),:);
best_options = best_options(opt(1):size(best_options,1),:);
i = i+1;
end
end

%Finds the matrix good_obj_func2, containing all
%architectures which have objective function values
%within 97.5% of the "optimal", but vary from this
%"optimal" solution by the number of at least one
%type of spacecraft.
good_options= data(find(data(:,1)>=0.975*max(data(:,1))),:);
good_options = -1*sortrows(-1*good_options,1);
good_obj_func2(1,:) = good_options(1,:);
opt = 1;
i = 2;
while isempty(opt)~=1
opt = find(good_options(:,4)~=good_options(1,4)|...

good_options(:,3)~=good_options(1,3)|good_options(:,2)~=good
_options(1,2));
if isempty(opt)~=1
good_obj_func2(i,:) = good_options(opt(1),:);
good_options = good_options(opt(1):size(good_options,1),:);
i = i+1;
end
end

%Finds all architectures in terms of just the number of
%each type of spacecraft that have been tested.  Used
%to plot design space covered.
test_options = data;
test_options = -1*sortrows(-1*test_options,1);
tested(1,:) = test_options(1,:);
opt = 1;
i = 2;
%while i<10
while isempty(opt)~=1
opt = find(test_options(:,4)~=test_options(1,4)|...
```

```
test_options(:,3)~=test_options(1,3)|test_options(:,2)~=test
_options(1,2));
if isempty(opt)~=1
tested(i,:) = test_options(opt(1),:);
test_options = test_options(opt(1):size(test_options,1),:);
i = i+1;
end
end
```

A.8 "sensitivity.m"

```
%This function calculates the sensitivity of each life cycle
%metric to each user defined parameter, using finite
%differences.  The parameters to be perturbed (tested) can
%be set in the inputs.m file.  The user can also set
%the number of perturbations and amount of perturbation in
%this file.  The data for all tests is returned in the
%architecture matrix. Note that in order to use this
%file, the ARCHITECTURE_MATRIX in the inputs.m file must
%contain only one architecture.
%This will be the only architecture tested for sensitivity.
%This architecture needs to be changed in the inputs.m file
%to find the sensitivity of other architectures.
%-------------------------------------------------------------
%-------------------------------------------------------------
%INPUTS
%No function inputs.  See inputs.m file.
%Parameters to vary, amount of variation, and number of
%variation points can all be set in this file.
%
%OUTPUTS -
%Lifecycle metrics (NoI, CpI, Reliability, and "score" for
%each change in each parameter.
%-------------------------------------------------------------
%-------------------------------------------------------------

clear

inputs

mo = COMBINER_OPTICS_FAILURE_RATE;
%Failure rate of combiner optics in months^-1
lo = COLLECTOR_OPTICS_FAILURE_RATE;
%Failure rate of collector optics in months^-1
m = COMBINER_BUS_FAILURE_RATE;
%Failure rate of combiner bus in months^-1
l = COLLECTOR_BUS_FAILURE_RATE;
%Failure rate of collector bus in months^-1
```

```
d = DUAL_BUS_FAILURE_RATE;
%Failure rate of dual functioning bus in months^-1
life = MISSION_DESIGN_LIFETIME;
%Mission design life in months
delt = TIME_STEP;
%Time step of one day for five years expressed in months
N = NUMBER_OF_DIFFERENCES;
%Number of independant differences (number of pixels/2)
S = SCALE_FACTOR_FOR_INCREASE;
%Scale factor for increased reliability
Co = TIME_PER_CONFIGURATION;
%Time in months needed in each configuration
Ot = OVERHEAD_TIME_PER_IMAGE;
%Overhead time per image in months
mo_tfu     =     COMBINER_OPTICS_THEORETICAL_FIRST_UNIT_COST;
%Cost of combiner optics theoretical first unit cost in $M
lo_tfu     =     COLLECTOR_OPTICS_THEORETICAL_FIRST_UNIT_COST;
%Cost of collector optics theoretical first unit cost in $M
mb_tfu      =      COMBINER_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of combiner bus theoretical first unit cost in $M
lb_tfu      =      COLLECTOR_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of collector bus theoretical first unit cost in $M
db_tfu = DUAL_BUS_THEORETICAL_FIRST_UNIT_COST;
%Cost of dual functioning bus theoretical first unit cost in
%$M
ops = OPERATIONS_COST_PER_BASELINE;
%Operations cost per month per baseline (%M/month)
s_lc = LEARNING_CURVE_SLOPE;
%Learning curve slope percentage (see SMAD for recommended
%values)
w_cpi = WEIGHTING_FOR_CPI;
%Weighting for cpi used in objective function
avg_cpi = AVERAGE_VALUE_FOR_CPI;
%Average value for cpi used in objective function
w_noi = WEIGHTING_FOR_NOI;
%Weighting for noi used in objective function
avg_noi = AVERAGE_VALUE_FOR_NOI;
%Average value for noi used in objective function
w_rel = WEIGHTING_FOR_REL;
%Weighting for rel used in objective function
avg_rel = AVERAGE_VALUE_FOR_REL;
%Average value for rel used in objective function
archs = ARCHITECTURE_MATRIX;
%Matrix of architectures to be evaluated.
num_points = NUMBER_OF_SENSITIVITY_POINTS;
%Number of points used to check the sensitivity to each
%variable
delta = CHANGE_IN_SENSITIVITY_POINTS;
%Change between each point for each variable for sensitivity
%analysis (given in decimals, not percentages - i.e. .1 not
%10)
```

```
%List of which parameters to calculate senstivity to.
%Vector of ones and zeros where 1 = calculate sensitivity.
in_vector_change = PARAMETERS_TO_CALCULATE_SENSITIVITY_TO;


num = size(archs,1);
if num ~= 1
    error('Multiple architectures in Architecture matrix')
end

%in_vector is the vector of inputs to calculate sensitivity
%to
in_vector_original = [mo lo m l d life S N Co Ot mo_tfu
lo_tfu mb_tfu lb_tfu db_tfu ops s_lc w_cpi avg_cpi w_noi
avg_noi w_rel avg_rel];
if length(in_vector_change)~=length(in_vector_original)
      error('In_vector_change is not the same length as
in_vector_original')
end

architecture2 = zeros(1,6);
index = 1;
while index <= length(in_vector_original)
    if in_vector_change(index) == 1
        i = -(delta*(num_points-1))/2;
%i defines the percentage taken off nominal value
%of the index variable for this run
        j = 1;
%j defines how many points have been evaluated (including
%this one) for this variable
        k = 1;
%k is used to keep track of the column to put the results
%into the architecture matrix.  5 results are stored for each
%run.
        while i <= delta*(num_points-1)/2
            in_vector = in_vector_original;
            in_vector(index) = in_vector(index)*(1+i);


%Clear all variables which are not constants, num, or being
%stored for comparison


clear P P0 t col com dual A n n1 Nc Tc Ti Ci a I M NoI ci R
Rel Cpi

            %Get architecture specific inputs
            dual = archs(1,1);
            com = archs(1,2);
            col = archs(1,3);
```

```
            %Call subroutine "DV_to_J_sens.m" to get the
          %number of images collected
            %Note that all inputs to which sensitivity will
          %be calculated are called
            %from this file and not from inputs.m

          x = [dual com col 0 0 0];
            [J, NoI, a, Rel, Cpi] = DV_to_J_sens(x,in_vector);

           %Store the information for each architecture to
        %graph at the end
           Number_of_Images(index,j) = NoI;
           Cost_per_Image(index,j) = Cpi;
           Reliabilities(index,j) = Rel;
             in_changes(index,j) = in_vector(index);
           Performance(index,j) = J;
             %This is the relevant information for each method
             %to allow graphing of # of images vs.
          %# of collectors, etc
          architecture(index,k:k+5) =
[(1+i)*100 in_vector(index) NoI/1000 Cpi Rel J];
                    architecture2(size(architecture2,1)+1,:) =
[(1+i)*100 in_vector(index) NoI/1000 Cpi Rel J];

            i = i + delta;   %Go on to next value for variable
             j = j+1;
             k = k+6;
          end
      end  %Stop if done with variable

      index = index + 1;
end

architecture
```

Appendix B

MODIFICATIONS TO *GAOT* TOOLBOX

The publicly accessible Matlab toolbox *GAOT* was used to implement the genetic algorithm optimization scheme discussed in Chapter 5. The first three design variables in the design vector for this problem contain integer constraints (one cannot buy half a spacecraft) and therefore, to accommodate this constraint, a modified binary scheme was chosen for the encoding of the design vectors to "genes." Several modifications were necessary in order to get the binary encoding aspects of toolbox algorithm working correctly. Additional modifications were made to this binary encoding to ensure the integer constraints needed for this problem were met. Specifically, a line was added which rounded the float number to the nearest integer after changing from binary to float formats. Once these modifications were made, they were tested by running several iterations through a function which chose a random integer design vector, encoded it using the binary encoding scheme, decoded it, and checked the original vector against the new decoded vector. These tests led to confidence in the modified encoding scheme. The full list of modifications to the *GAOT* toolbox is shown below.

- Changed first line of *"f2b.m"* to second line.
- Changed line 152 of *"ga.m"* to:
  ```
  bits=calcbits(bounds,opts(1)*ones(1,size(bounds,1)));
  ```
- Changed lines 96-97 and 104-105 of *"ga.m"* to
  ```
  elstr=['x=b2f(endPop(j,1:numVar),bounds,bits);end-
  Pop(j,xZomeLength)=',...
  evalFN ';'];
  ```

- Changed line 74 in *"initializega.m"* to:
```
bits=calcbits(bounds,options(1)*ones(1,size(bounds,1)));
```
- Added ";" to line 256 of *"ga.m"*
- Added while loop below line 256 of *"ga.m"*
```
while isempty(cp)
          cp=find(rand(popSize,1)<xOverOps(i,1)==1);
          if rem(size(cp,1),2); cp=cp(1:(size(cp,1)-1)); end
end
```
- Added rounding after line 35 of *"b2f.m"*. Note that this line should only be added to enforce an integer constraint.
```
fval(i)=round(fval(i));
```
- Changed line 29 on *"tournSelect.m"* to (Note: this change is necessary in order to get tournament selection working, and has nothing to do with binary encryption):
```
tournSize=ceil(size(oldPop,1)/4);
```

Appendix C

OPTIMIZATION RESULTS

Chapter 5 discussed two different heuristic optimization algorithms and their results when applied to the SSI conceptual design problem - simulated annealing and genetic algorithms. The results from these optimization algorithms were summarized in Chapter 5. The full results are listed in this appendix, first for simulated annealing and then for genetic algorithms.

C.1 Simulated Annealing

TABLE C.1 Simulated annealing optimization set-up

Budget	$360M
Iterations	1500
Steps	300
Delta Guess	0.1
dual bounds	[0:1:5]
com bounds	[0:1:6]
col bounds	[0:1:12]
Xmo bounds	[0:5:100]
Xlo bounds	[0:5:100]
Xb bounds	[0:5:100]

TABLE C.2 "Best" architecture returned by the simulated annealing optimization algorithm.

"Best" Architecture
0 dual functioning spacecraft
2 combining spacecraft
5 collecting spacecraft
$10M on improving combining optics reliability
$5M on improving collecting optics reliability
$25M on improving bus reliability

TABLE C.3 Architectures returned by the simulated annealing algorithm with objective values within 97.5% of the "optimal" but which also vary from the "optimal" by the number of at least one type of spacecraft. Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2937	0	2	5	10	5	25
1.2876	0	2	4	10	20	25
1.2872	0	2	4	10	15	35
1.2863	0	2	4	15	20	25
1.2856	0	2	4	10	25	35
1.2833	0	2	4	10	25	40
1.2826	0	2	4	10	30	35
1.2821	0	2	4	10	10	40

TABLE C.4 Architectures returned by the simulated annealing algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2937	0	2	5	10	5	25
1.2909	0	2	5	10	15	35
1.2903	0	2	5	10	20	25
1.2892	0	2	5	10	0	30
1.289	0	2	5	15	5	40
1.2889	0	2	5	10	0	25
1.2886	0	2	5	15	0	30
1.2876	0	2	5	5	5	40
1.2876	0	2	4	10	20	25
1.2874	0	2	4	10	20	35
1.2873	0	2	5	5	20	25
1.2872	0	2	4	10	15	35
1.2866	0	2	5	10	25	25
1.2864	0	2	5	10	0	20
1.2863	0	2	4	15	20	25
1.2863	0	2	5	5	0	35
1.2858	0	2	5	5	15	40
1.2856	0	2	4	10	25	35
1.2855	0	2	4	15	15	35
1.285	0	2	4	10	20	40
1.284	0	2	4	5	20	40
1.2837	0	2	4	10	10	25
1.2836	0	2	5	25	5	40
1.2833	0	2	4	10	25	40
1.2829	0	2	4	20	20	35
1.2827	0	2	5	5	20	40
1.2826	0	2	4	10	30	35
1.2823	0	2	5	10	30	25
1.2821	0	2	4	10	10	40
1.2809	0	2	4	5	15	45

C.2 Genetic Algorithm

TABLE C.5 *Genetic algorithm optimization set-up*

Budget	$360M
Generations	60
Population Size	50
Crossover rate	0.6
Crossover selection	Tournament
Mutation rate	0.1
Mutation type	Binary
dual bounds	[0:1:5]
com bounds	[0:1:6]
col bounds	[0:1:12]
Xmo bounds	[0:1:100]
Xlo bounds	[0:1:100]
Xb bounds	[0:1:100]

TABLE C.6 "Best" architecture returned by the genetic algorithm.

"Best" Architecture
0 dual functioning spacecraft
2 combining spacecraft
5 collecting spacecraft
$12M on improving combining optics reliability
$9M on improving collecting optics reliability
$25M on improving bus reliability

TABLE C.7 Architectures returned by the genetic algorithm with objective values within 97.5% of the "optimal" but which also vary from the "optimal" by the number of at least one type of spacecraft. Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2949	0	2	5	12	9	25
1.2868	1	1	4	12	13	30
1.2867	1	1	4	12	13	31
1.2865	1	1	4	11	13	31
1.2863	1	1	4	10	13	30
1.2861	1	1	4	12	14	33
1.286	1	1	4	12	13	34
1.2858	1	1	4	16	17	25
1.2853	1	1	4	12	13	36
1.2851	1	1	4	12	17	33
1.2849	1	1	4	11	18	31
1.2845	1	1	4	7	14	28
1.2844	1	1	4	11	18	33
1.2842	1	1	4	10	19	26
1.284	1	1	4	9	13	37
1.2838	1	1	4	6	13	30
1.2837	1	1	4	7	15	33
1.2836	1	1	4	6	14	30
1.2835	1	1	4	12	19	35
1.283	1	1	4	10	21	31
1.2829	1	1	4	5	13	29
1.2827	1	1	4	11	13	41
1.282	1	1	4	12	16	41
1.2814	1	1	4	5	14	36
1.2812	1	1	4	5	17	33
1.281	1	1	4	21	3	31
1.2807	1	1	4	3	10	30
1.2801	1	1	4	4	18	30
1.2799	1	1	4	12	1	31
1.2798	1	1	4	7	17	19
1.2793	1	1	4	18	1	31
1.2789	1	1	4	5	13	42
1.2788	1	1	4	12	1	37
1.2783	1	1	4	2	14	34
1.278	1	1	4	6	24	33
1.2777	1	1	4	12	6	48

TABLE C.7 Architectures returned by the genetic algorithm with objective values within 97.5% of the "optimal" but which also vary from the "optimal" by the number of at least one type of spacecraft. Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2773	1	1	4	9	0	31
1.2771	1	1	4	5	2	36
1.277	1	1	4	6	7	46
1.2769	1	1	4	1	14	33
1.2762	1	1	4	5	1	30
1.2744	1	1	4	5	17	47
1.2731	1	1	4	12	1	48
1.2725	1	1	4	33	3	25
1.2723	1	1	3	12	21	33
1.2719	1	1	4	3	22	43
1.2718	1	1	3	10	23	34
1.2715	1	1	4	12	15	11
1.2713	1	1	3	11	19	30
1.2709	1	1	3	12	16	34
1.2702	1	1	4	12	21	11
1.2702	0	3	5	3	3	25
1.27	1	1	3	12	17	28
1.2693	1	1	3	12	14	32
1.2693	0	3	5	5	3	25
1.2692	1	1	4	6	13	12
1.2691	1	1	3	8	16	31
1.2681	1	1	3	11	18	45
1.2681	1	1	4	5	14	12
1.2678	1	1	3	23	16	33
1.2675	1	1	3	4	21	34
1.2672	1	1	3	11	14	27
1.2672	2	0	4	12	1	35
1.2669	1	1	4	35	0	30
1.2668	1	1	3	11	13	28
1.2662	1	1	3	21	13	31
1.2657	1	1	3	5	14	37
1.2657	0	3	5	6	10	9
1.2656	1	1	3	8	15	45
1.2652	2	0	4	5	17	29
1.2652	1	1	4	3	15	12
1.265	2	1	3	9	10	18

TABLE C.7 Architectures returned by the genetic algorithm with objective values within 97.5% of the "optimal" but which also vary from the "optimal" by the number of at least one type of spacecraft. Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2649	1	1	3	12	38	33
1.2645	0	3	5	8	1	21
1.264	1	1	3	12	38	28
1.264	1	2	4	9	9	14
1.2632	0	2	6	1	3	19
1.2626	1	1	3	25	16	24

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2949	0	2	5	12	9	25
1.2949	0	2	5	12	10	27
1.2949	0	2	5	12	8	27
1.2949	0	2	5	12	10	25
1.2949	0	2	5	11	9	25
1.2949	0	2	5	11	10	27
1.2949	0	2	5	12	8	28
1.2948	0	2	5	12	10	28
1.2948	0	2	5	12	11	25
1.2948	0	2	5	11	8	25
1.2948	0	2	5	12	11	27
1.2947	0	2	5	14	11	25
1.2947	0	2	5	10	10	27
1.2947	0	2	5	12	7	25
1.2946	0	2	5	12	9	23
1.2946	0	2	5	12	12	27
1.2945	0	2	5	9	9	27
1.2945	0	2	5	12	12	24
1.2945	0	2	5	9	8	27
1.2945	0	2	5	12	12	28
1.2945	0	2	5	12	6	27
1.2944	0	2	5	9	10	27
1.2944	0	2	5	9	9	25
1.2944	0	2	5	9	10	25
1.2944	0	2	5	16	11	25
1.2943	0	2	5	9	11	26
1.2943	0	2	5	9	11	27
1.2942	0	2	5	9	7	25
1.2942	0	2	5	12	10	31
1.2942	0	2	5	11	10	22
1.2942	0	2	5	11	7	23
1.2941	0	2	5	12	6	30
1.2941	0	2	5	17	9	25
1.2941	0	2	5	11	6	30
1.2941	0	2	5	11	8	22
1.2941	0	2	5	8	9	25

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.294	0	2	5	12	5	27
1.294	0	2	5	17	9	24
1.2939	0	2	5	12	6	23
1.2937	0	2	5	12	5	30
1.2937	0	2	5	9	5	27
1.2936	0	2	5	9	5	26
1.2935	0	2	5	9	10	32
1.2935	0	2	5	15	5	25
1.2935	0	2	5	12	10	20
1.2935	0	2	5	12	4	27
1.2935	0	2	5	11	4	27
1.2934	0	2	5	8	7	23
1.2934	0	2	5	7	11	27
1.2934	0	2	5	10	4	27
1.2933	0	2	5	7	11	25
1.2933	0	2	5	9	8	33
1.2933	0	2	5	12	6	33
1.2933	0	2	5	12	12	20
1.2933	0	2	5	12	9	34
1.2932	0	2	5	16	5	25
1.2932	0	2	5	7	11	29
1.2932	0	2	5	12	4	30
1.2932	0	2	5	9	10	33
1.2929	0	2	5	9	4	30
1.2929	0	2	5	6	9	25
1.2929	0	2	5	9	14	30
1.2928	0	2	5	12	9	19
1.2928	0	2	5	12	4	23
1.2927	0	2	5	7	8	22
1.2927	0	2	5	12	12	19
1.2927	0	2	5	11	3	27
1.2927	0	2	5	16	16	27
1.2927	0	2	5	12	3	27
1.2927	0	2	5	12	8	19
1.2925	0	2	5	12	3	25
1.2925	0	2	5	6	9	23

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2925	0	2	5	18	5	25
1.2924	0	2	5	7	4	27
1.2923	0	2	5	21	9	27
1.2923	0	2	5	6	7	23
1.2923	0	2	5	12	3	24
1.2923	0	2	5	21	12	25
1.2923	0	2	5	6	12	24
1.2922	0	2	5	9	4	33
1.2922	0	2	5	9	9	19
1.2922	0	2	5	5	9	25
1.2921	0	2	5	6	9	22
1.2921	0	2	5	7	4	31
1.292	0	2	5	12	17	30
1.2919	0	2	5	21	9	22
1.2918	0	2	5	12	9	37
1.2918	0	2	5	12	3	33
1.2917	0	2	5	9	17	28
1.2917	0	2	5	5	9	23
1.2916	0	2	5	6	6	22
1.2916	0	2	5	5	12	25
1.2916	0	2	5	18	4	30
1.2914	0	2	5	4	8	27
1.2914	0	2	5	4	7	27
1.2913	0	2	5	5	9	22
1.2913	0	2	5	9	17	30
1.2913	0	2	5	12	9	17
1.2912	0	2	5	4	9	25
1.2912	0	2	5	23	9	25
1.2912	0	2	5	23	10	27
1.2911	0	2	5	11	9	17
1.2909	0	2	5	11	19	29
1.2908	0	2	5	12	4	37
1.2907	0	2	5	18	11	36
1.2907	0	2	5	9	3	21
1.2907	0	2	5	24	11	25
1.2906	0	2	5	12	10	39

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2906	0	2	5	24	9	25
1.2905	0	2	5	12	1	30
1.2905	0	2	5	24	9	27
1.2904	0	2	5	24	8	25
1.2904	0	2	5	17	2	25
1.2903	0	2	5	11	19	31
1.2903	0	2	5	6	2	27
1.2902	0	2	5	12	8	40
1.2902	0	2	5	11	9	16
1.2901	0	2	5	9	18	32
1.2901	0	2	5	3	6	28
1.2901	0	2	5	12	21	25
1.29	0	2	5	21	4	30
1.2899	0	2	5	12	21	23
1.2897	0	2	5	11	4	39
1.2896	0	2	5	12	17	36
1.2896	0	2	5	12	1	34
1.2895	0	2	5	18	4	36
1.2895	0	2	5	22	9	18
1.2893	0	2	5	3	4	27
1.2892	0	2	5	24	6	30
1.2892	0	2	5	24	11	19
1.2891	0	2	5	11	22	27
1.2891	0	2	5	12	3	39
1.2888	0	2	5	8	10	41
1.2887	0	2	5	24	5	30
1.2885	0	2	5	6	18	32
1.2884	0	2	5	2	9	23
1.2883	0	2	5	2	5	25
1.2879	0	2	5	12	3	17
1.2879	0	2	5	12	4	42
1.2878	0	2	5	11	23	30
1.2877	0	2	5	12	23	20
1.2876	0	2	5	6	17	36
1.2871	0	2	5	9	9	14
1.2871	0	2	5	5	20	29

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2871	0	2	5	6	17	37
1.2871	0	2	5	1	11	25
1.2869	0	2	5	9	24	27
1.2868	1	1	4	12	13	30
1.2868	0	2	5	29	9	23
1.2867	1	1	4	12	13	31
1.2866	1	1	4	11	13	29
1.2866	1	1	4	12	14	30
1.2866	1	1	4	18	13	27
1.2866	1	1	4	11	13	30
1.2866	0	2	5	8	11	44
1.2865	0	2	5	6	10	43
1.2865	0	2	5	22	8	40
1.2865	1	1	4	11	13	31
1.2864	1	1	4	12	15	30
1.2863	0	2	5	7	19	17
1.2863	1	1	4	10	13	30
1.2863	1	1	4	12	13	33
1.2862	1	1	4	12	15	31
1.2862	1	1	4	11	13	25
1.2862	1	1	4	12	16	28
1.2862	0	2	5	30	9	25
1.2861	1	1	4	12	14	33
1.286	0	2	5	6	4	43
1.286	1	1	4	12	13	34
1.2859	1	1	4	11	15	25
1.2858	1	1	4	9	13	30
1.2858	0	2	5	9	24	32
1.2858	1	1	4	16	17	25
1.2858	1	1	4	12	15	33
1.2857	1	1	4	9	13	31
1.2857	1	1	4	15	18	27
1.2856	1	1	4	12	13	35
1.2856	1	1	4	11	15	33
1.2855	1	1	4	12	15	34
1.2853	1	1	4	11	15	34

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2853	0	2	5	16	12	12
1.2853	1	1	4	12	13	36
1.2853	1	1	4	10	15	33
1.2852	1	1	4	8	13	31
1.2851	0	2	5	29	8	33
1.2851	1	1	4	12	17	33
1.285	1	1	4	15	19	30
1.285	1	1	4	12	19	28
1.2849	1	1	4	21	15	31
1.2849	0	2	5	12	9	12
1.2849	1	1	4	11	18	31
1.2849	1	1	4	12	13	37
1.2848	1	1	4	8	13	33
1.2848	1	1	4	12	19	30
1.2847	0	2	5	11	9	12
1.2846	0	2	5	12	28	25
1.2845	1	1	4	7	14	28
1.2844	1	1	4	12	20	27
1.2844	0	2	5	4	22	23
1.2844	1	1	4	11	18	33
1.2843	0	2	5	31	9	20
1.2842	1	1	4	10	19	26
1.2842	0	2	5	6	21	17
1.284	1	1	4	9	13	37
1.2839	1	1	4	8	13	36
1.2839	1	1	4	12	21	28
1.2838	0	2	5	3	17	38
1.2838	1	1	4	6	13	30
1.2838	1	1	4	11	16	37
1.2837	0	2	5	6	10	47
1.2837	1	1	4	7	15	33
1.2837	0	2	5	3	21	22
1.2836	1	1	4	6	14	30
1.2835	0	2	5	9	9	49
1.2835	1	1	4	12	19	35
1.2835	1	1	4	25	13	30

TABLE C.8 Architectures returned by the genetic algorithm with objective function values within 99% of "optimal". Note that the first architecture listed is the "optimal" architecture.

Objective Function	Duals	Combs	Cols	$ spent on comb. optics ($M)	$ spent on col. optics ($M)	$ spent on bus ($M)
1.2834	1	1	4	25	14	30
1.2834	1	1	4	6	15	30
1.2833	1	1	4	6	13	25
1.2832	1	1	4	6	14	33
1.2832	1	1	4	8	19	30
1.2832	1	1	4	25	15	30
1.2831	1	1	4	12	21	33
1.283	0	2	5	11	9	50
1.283	1	1	4	10	21	31
1.2829	0	2	5	12	7	50
1.2829	1	1	4	5	13	29
1.2829	1	1	4	12	13	41
1.2828	1	1	4	5	13	31
1.2828	1	1	4	17	23	25
1.2828	1	1	4	12	21	34
1.2827	0	2	5	9	9	50
1.2827	1	1	4	11	13	41
1.2827	1	1	4	5	14	29
1.2827	1	1	4	5	14	30
1.2826	1	1	4	12	3	32
1.2825	1	1	4	5	13	33
1.2824	1	1	4	12	21	35
1.2824	1	1	4	8	19	34
1.2823	1	1	4	12	13	42
1.2823	1	1	4	6	14	36
1.2823	0	2	5	10	30	25
1.2821	0	2	5	29	3	34
1.282	1	1	4	12	16	41

Appendix D

SENSITIVITY ANALYSIS RESULTS

The analysis presented above is dependant upon 23 user defined inputs, or parameters, which are listed in Table 5.8. These parameters are constant for all architectures being evaluated, but can affect different architectures in different ways. The sensitivity of the solutions returned by all of the analyses discussed above to all of these parameters is crucial. Therefore a sensitivity analysis tool has been developed to find the sensitivity of the models to these parameters. Please see Chapter 5 for a full discussion of this tool. This appendix will present the full results of the case study of the sensitivity of two architectures. The first architecture consists of no dual functioning spacecraft, two combining spacecraft, five collecting spacecraft, $12M spent to improve the reliability of the combining optics, $9M spent to improve the reliability of the collecting optics, and $25M spent to improve the reliability of the bus. The second architecture consists of one dual functioning spacecraft, one combining spacecraft, four collecting spacecraft, $12M spent to improve the reliability of the combining optics, $13M spent to improve the reliability of the collecting optics, and $30M spent to improve the reliability of the bus. The results are shown for a sensitivity analysis of the architectures defined only the number of each type of spacecraft first, followed by the results of a sensitivity analysis when the architectures are defined with the money spent to improve the reliability of components included.

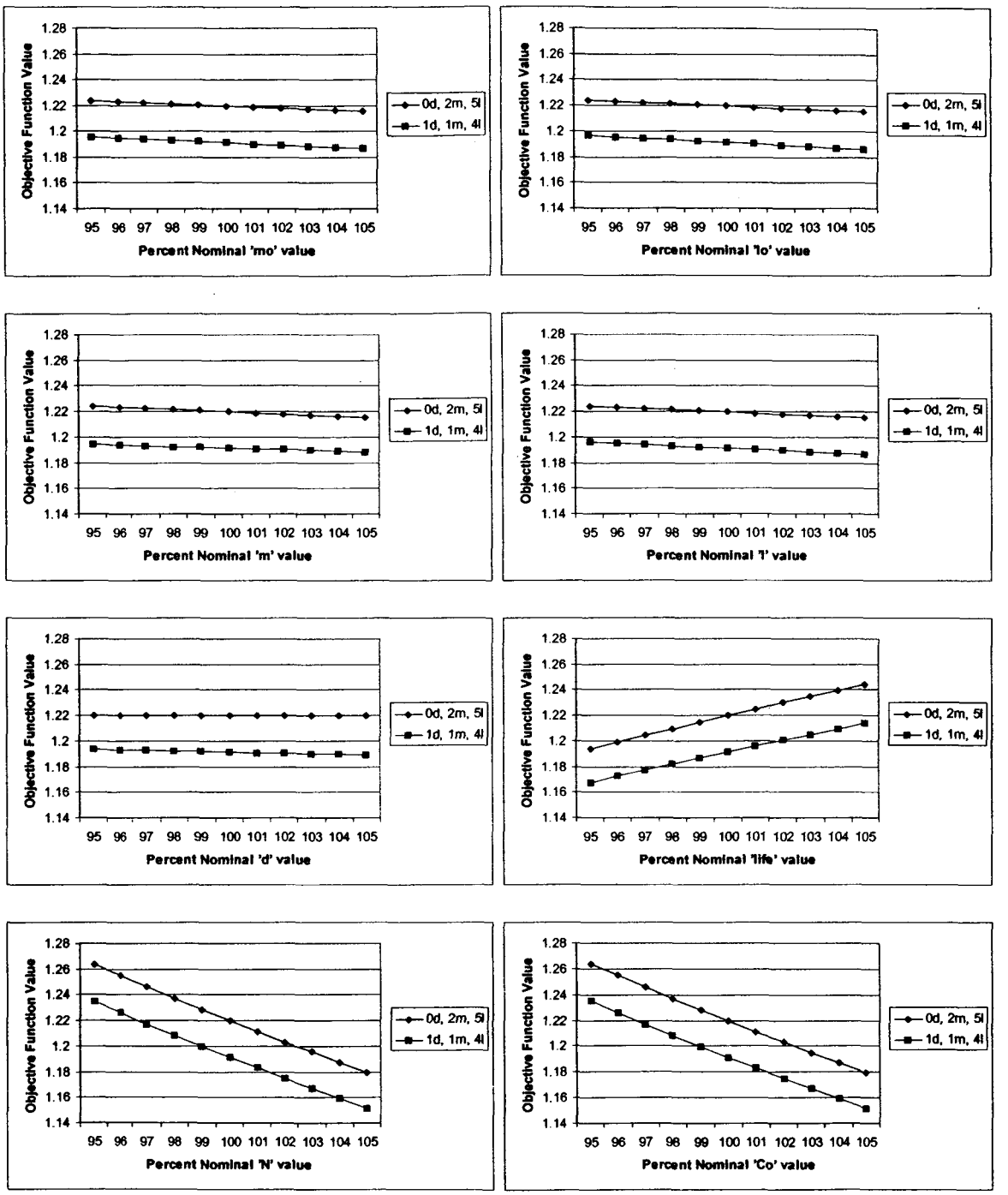


Figure D.1 Sensitivity analysis of the architectures given above, defined by only the number of each type of spacecraft.

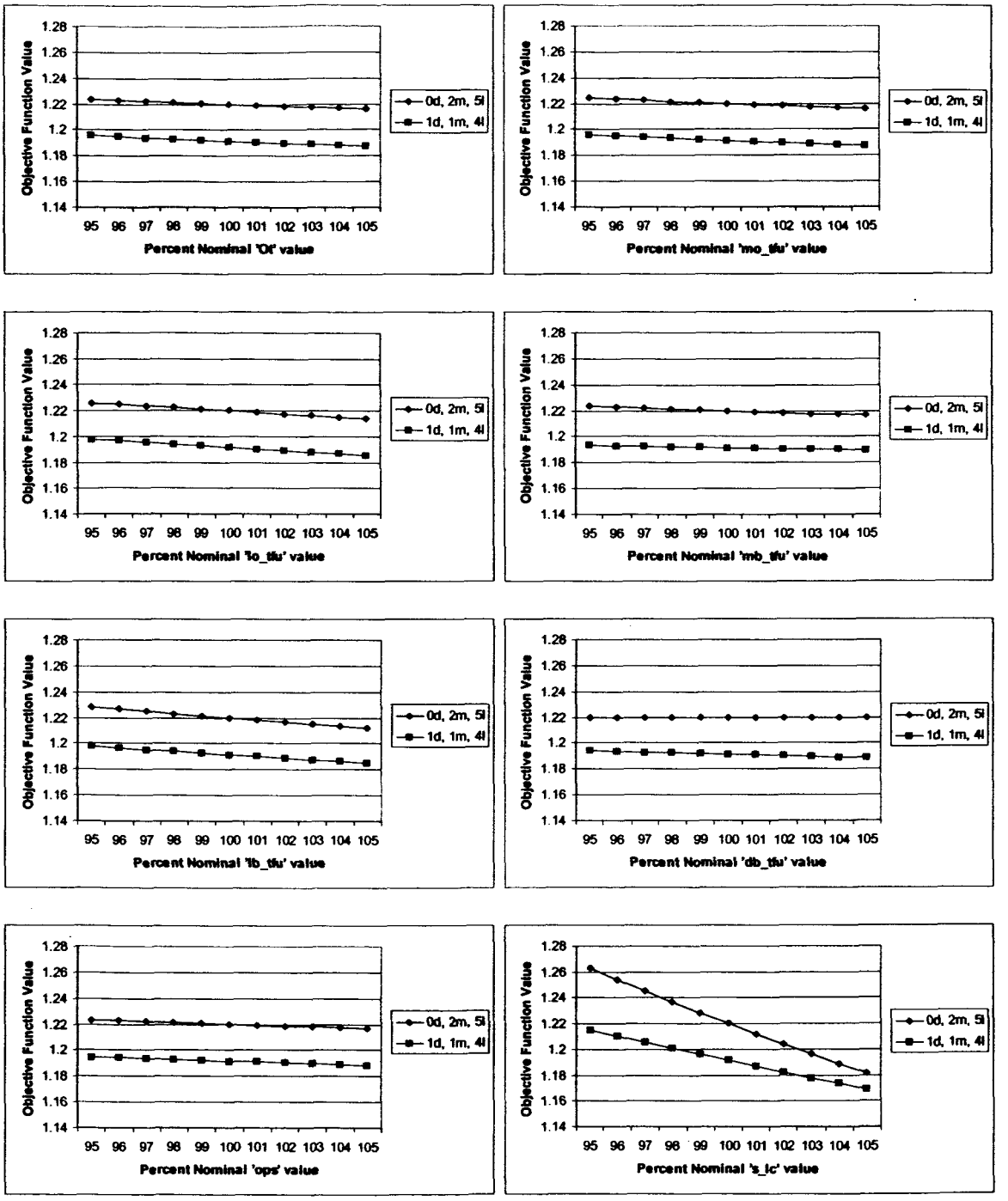


Figure D.1 Sensitivity analysis of the architectures given above, defined by only the number of each type of spacecraft.

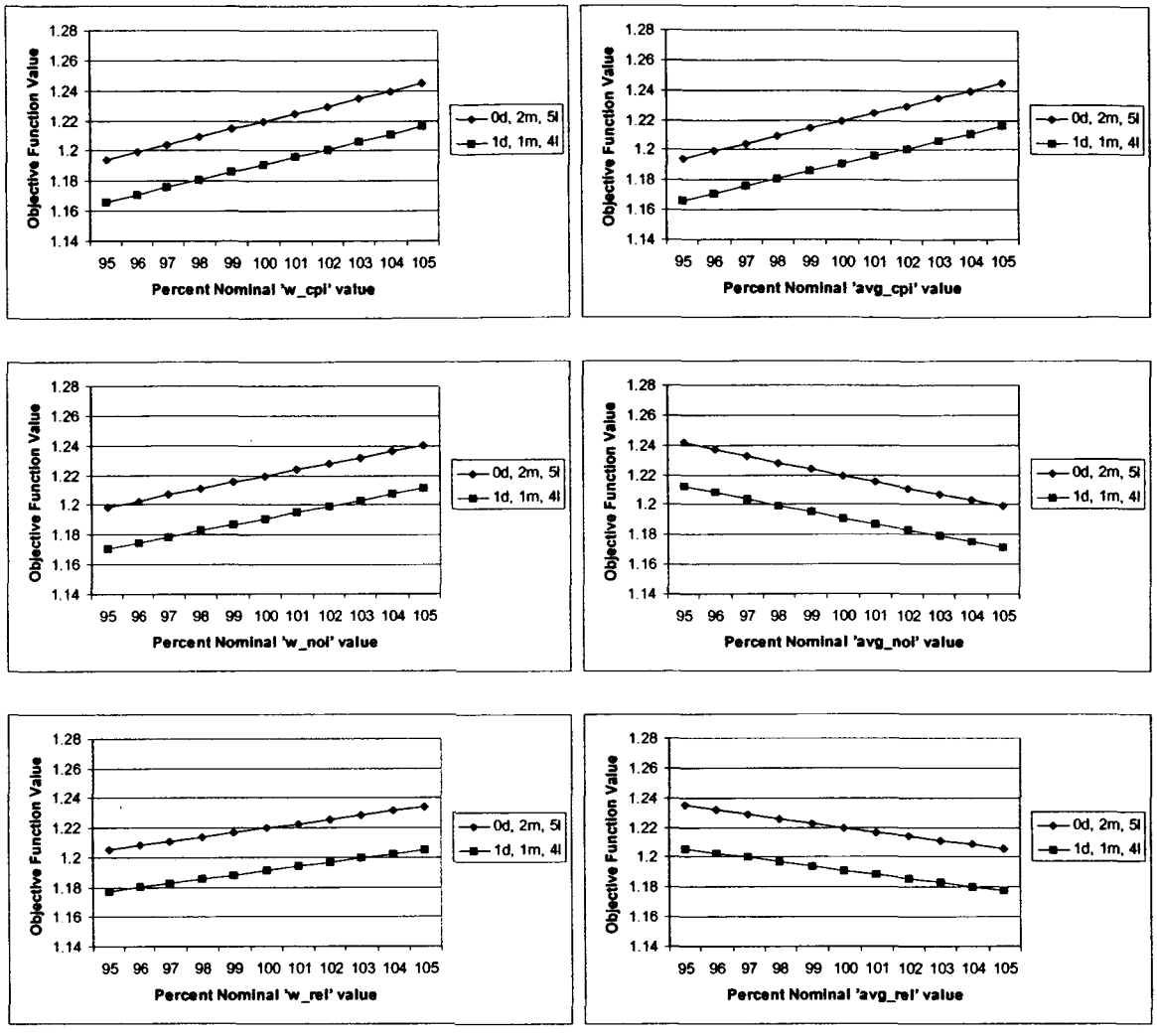


Figure D.1 Sensitivity analysis of the architectures given above, defined by only the number of each type of spacecraft.

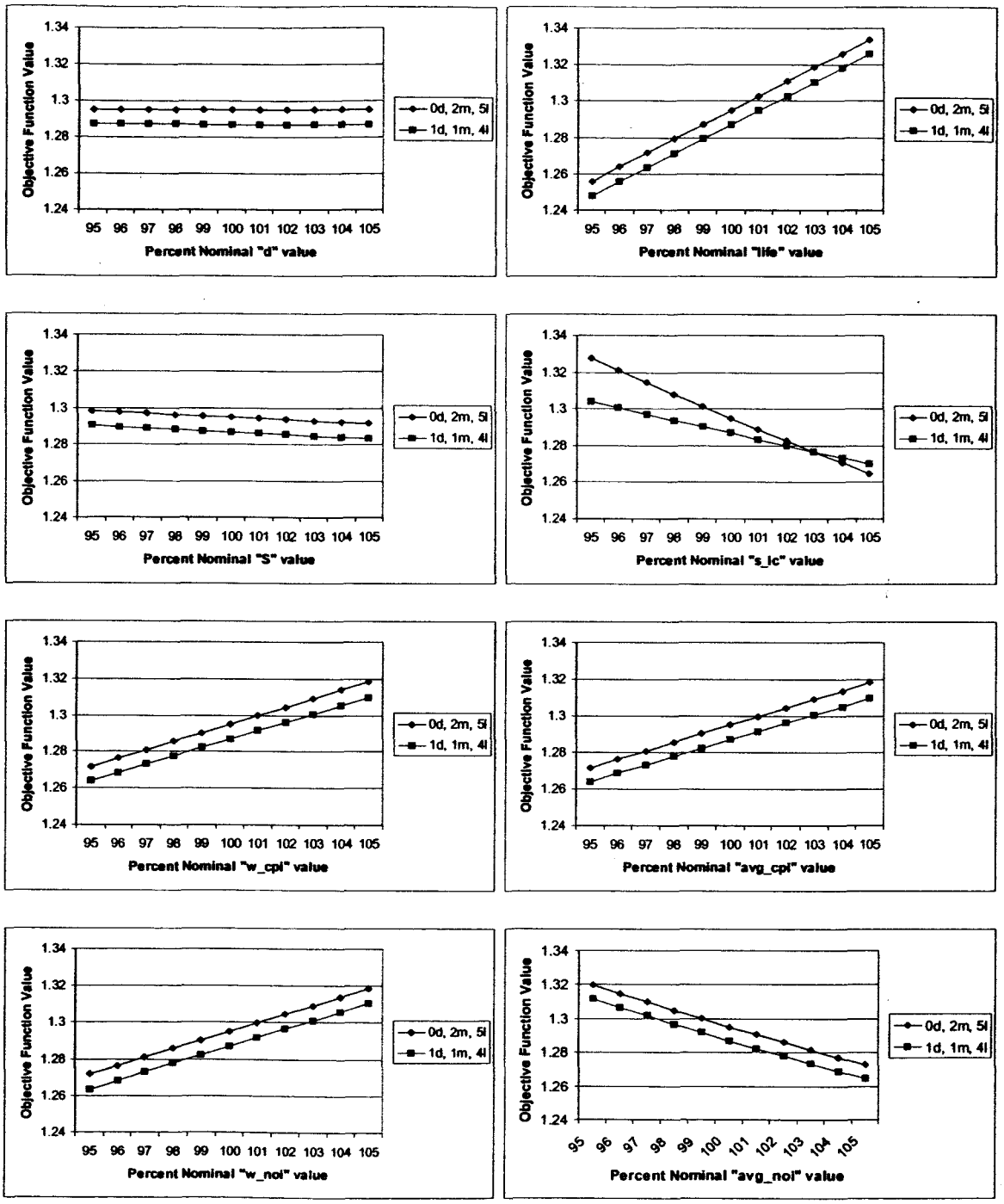


Figure D.2 Sensitivity analysis of the architectures given above, defined by both the number of each type of spacecraft and the money spent to improve component reliabilities.

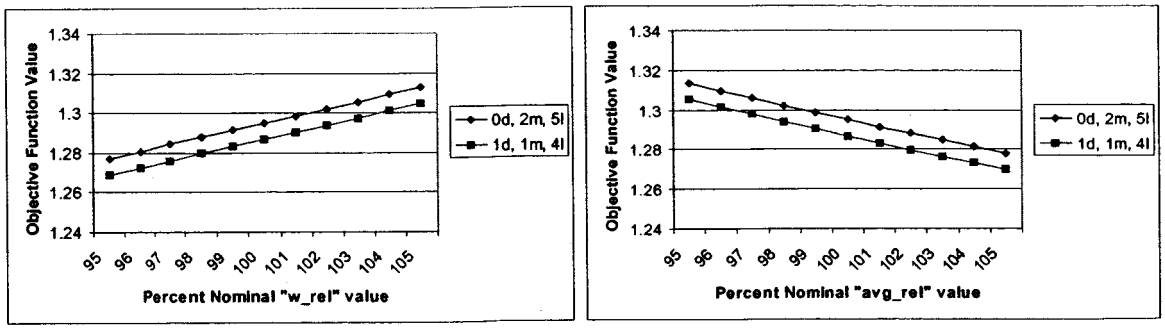


Figure D.2 Sensitivity analysis of the architectures given above, defined by both the number of each type of spacecraft and the money spent to improve component reliabilities.

Appendix E

MATLAB TOOLBOX DESCRIPTION

A Matlab toolbox containing all the tools and models previously described has been assembled. Table E.1 lists all the files found in the toolbox, with a brief description of what the file does and it's inputs and outputs for each. Please note that Table E.1 does not include all the files contained in the GAOT toolbox even though these files are included in the Reliability and Productivity toolbox. Table Y lists all the user-defined input variables, including a description, the units used, and the default value for all variables.

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
Modeling Tools			
inputs.m	Contains all user defined inputs, or parameters. Called in other functions to set the values of these parameters.	N/A	N/A
DV_to_J.m	Finds the objective function, number of images, reliability, and cost per images of a given architecture.	Design vector (consists of the number of dual functioning, combining, and collecting spacecraft respectively, and the money spent to improve combining optics, collecting optics, and the bus in $M), *inputs.m* file	Number of Images (NoI), reliability, cost per image (CpI), A-matrix, and objective function
state.m	Recursive function which generates the state-transition matrix for a given system.	Current state-transition matrix, matrix with previously defined states, last state called, number of functioning spacecraft of each type, and failure rates of all components	State-transition matrix and matrix of state definitions
cost_model.m	Finds the total life-cycle cost of the given system.	Number of each type of spacecraft, money spent to improve reliabilities of each component, state-transition matrix, number of baselines in each state, and *inputs.m* file	Cost of system in $M

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
auto_a.m	Sets up initial conditions and calls *state.m* to automatically generate state-transition matrix.	Number of each type of spacecraft, failure rates of each component, and *inputs.m* file	State-transition matrix and matrix of state definitions
DV_to_NoI.m	Finds the expected total number of images a given system will produce.	Design vector and *inputs.m* file	NoI
Symbolic_a_matrix.m	Finds the state-transition matrix of a system symbolically.	Number of each type of spacecraft	Matrix of state definitions, number of baselines in each state, and symbolic state-transition matrix
reliability_w_symbolic_a.m	Finds the reliability of a system from the symbolic state-transition matrix.	Number of each type of spacecraft, money spent to improve the reliability of each component, symbolic state-transition matrix, total system budget, matrix of state definitions, number of baselines in each state, and *inputs.m* file	Reliability
Comparison Tools			
arch_comparison.m	Compares user given architectures in terms of total performance. Spends no money on improving component reliabilities.	*inputs.m* file	NoI, reliability, CpI, and "score" for each architecture

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
arch_comparison_w_impr.m	Compares user given architectures in terms of total performance. Includes optimal division of money to improve component reliabilities.	*inputs.m* file	NoI, reliability, CpI, and "score" for each architecture
arch_comparison_full_x.m	Same as *arch_comparison.m* but with matrix of architectures from *inputs.m* file containing money spent on reliabilities of components.	*inputs.m* file	NoI, reliability, CpI, and "score" for each architecture
arch_comparison_inputs.m	Same as *arch_comparison.m* but with matrix of architectures to be evaluated entered as input to function and not from *inputs.m* file.	Matrix of architectures to be evaluated, *inputs.m* file	NoI, reliability, CpI, and "score" for each architecture
arch_comparison_w_impr_inputs.m	Same as *arch_comparison_w_impr.m* but with matrix of architectures to be evaluated entered as input to function and not from *inputs.m* file.	Matrix of architectures to be evaluated, *inputs.m* file	NoI, reliability, CpI, and "score" for each architecture
arch_comparison_full_x_inputs.m	Same as *arch_comparison_full_x.m* but with matrix of architectures to be evaluated entered as input to function and not from *inputs.m* file.	Matrix of architectures to be evaluated, *inputs.m* file	NoI, reliability, CpI, and "score" for each architecture

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
optim_reliability_w_test.m	Optimization program to find the optimal method of dividing money among components to improve reliabilities. Used in *arch_comparison_w_impr.m.*	Number of combining, collecting, and dual functioning spacecraft, method of finding total budget, total system budget or percent of original system cost, and *inputs.m* file	"Optimal" amount of money to spend on improving each component's reliability in $M, final failure rates of components, and "optimal" reliability
sim_annealing_money_initial.m	Simulated annealing program to find the "optimal" method of dividing money among components to improve reliabilities. Used in *arch_comparison_w_impr.m*	Number of combining, collecting, and dual functioning spacecraft, total system budget, initial design vector in terms of money spent on each component, and *inputs.m* file	"Optimal" amount of money to spent on improving each component's reliability in $M and "optimal" system reliability
sim_annealing_money_tune.m	Same as *sim_annealing_money_intial.m* but with certain parameters given as function inputs, not from *inputs.m* file. Used for tuning *sim_annealing_money_initial.m*	Number of combining, collecting, and dual functioning spacecraft, total system budget, initial design vector in terms of money spent on each component, initial guess at difference between two neighboring vectors reliabilities, number of iterations, and *inputs.m* file	"Optimal" division of money and reliability for each test

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
inverse_ reliability.m	Finds the inverse of the reliability of a given system. Used as the objective function in *sim_annealing_money_ initial.m.*	Money spent to improve reliability of each component, number of each type of spacecraft, total system budget, and *inputs.m* file	Inverse reliability
optim_ reliability_tune. m	Calls *sim_annealing_money_ tune.m* with different input values to compare parameter settings and tune the algorithm.	N/A	"Optimal" division of money and total system reliability for each test
optim_ reliability.m	Same as *optim_reliability_w_ test.m* but with no sanity-check built in to see if simulated annealing algorithm found a local optimum.	Number of combining, collecting, and dual functioning spacecraft, method of finding total budget, total system budget or percent of original system cost, and *inputs.m* file	"Optimal" amount of money to spend on improving each component's reliability in $M, final failure rates of components, and "optimal" reliability
design_space_ view.m	Shows multiple views of the design space of finding optimal way to divide money to improve reliability, using two of the three design variables for each view.	Number of each type of spacecraft, total system budget, and which variables to enumerate design space with	Plots of design space of two variables at a time
Simulated Annealing			

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
sim_annealing.m	Finds the "optimal" architecture, in terms of performance for a given budget, defined by the number of each type of spacecraft and the money spent to improve the reliability of each component.	*inputs.m* file	"Optimal" architecture and objective function, all architectures considered "good enough" (objective functions within 99% and 97.5% of the "optimal", with the latter required to have a different number of at least one type of spacecraft)
sim_annealing_tune.m	Same as *sim_annealing.m* but with certain parameters given as function inputs instead of from *inputs.m* file. Used to tune simulated annealing algorithm.	Number of iterations, number of steps down in temperature, initial guess at difference in objective function between two neighboring design vectors, total system budget, and *inputs.m* file	"Optimal" architecture and objective function
sim_annealing_tune_data.m	Calls *sim_annealing_tune.m* with different input values to compare parameter settings and tune the algorithm.	N/A	"Optimal" architecture and objective function for each test
Genetic Algorithms			

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
J_GA.m	Calls genetic algorithm program to find the "optimal" architecture, in terms of performance for a given budget, defined by the number of each type of spacecraft and the money spent to improve the reliability of each component.	*inputs.m* file in *GA* folder	"Optimal" architecture and objective function, all architectures considered "good enough" (objective functions within 99% and 97.5% of the "optimal", with the latter required to have a different number of at least one type of spacecraft)
ga.m	Code to run the actual genetic algorithm.	See code	"Optimal" architecture and objective function
J_GA_tune.m	Same as *J_GA.m* but with certain parameters given as function inputs instead from *inputs.m*. Used to tune genetic algorithm.	Number of generations, number of individuals per generation, mutation rate, crossover rate, and *inputs.m* file in *GA* folder	"Optimal" architecture and objective function
ga_tune.m	Calls *J_GA_tune.m* with different input values to compare parameter settings and tune the algorithm.	N/A	"Optimal" architecture and objective function for each test
Sensitivity Analysis			

TABLE E.1 Descriptions, major inputs, and major outputs of files in the Reliability and Productivity Matlab toolbox.

Filename	Description	Major Inputs	Major Outputs
sensitivity.m	Finds the sensitivity of a given architecture to user defined parameters (does not include money to improve component reliabilities).	*inputs.m* file	Change in NoI, CpI, reliability, and objective function for each change in each parameter
sensitivity_full_x.m	Finds the sensitivity of a given architecture to user defined parameters (does include money to improve component reliabilities).	Design vector, *inputs.m* file	Change in NoI, CpI, reliability, and objective function for each change in each parameter
DV_to_J_sens.m	Same as *DV_to_J.m* but with parameter values as function inputs instead of from inputs.m file.	Design vector, parameter values	NoI, Reliability, CpI, A-matrix, and objective function
cost_model_sensitivity.m	Same as *cost_model.m* but with parameter values as function inputs instead of from *inputs.m* file.	Number of each type of spacecraft, money spent to improve reliabilities of each component, state-transition matrix, number of baselines in each state, and parameter values	Cost of system in $M

TABLE E.2 List of user-defined inputs for Reliability and Productivity toolbox

Global Variable Name	Description	Default Value	Units
Modeling Tools			
COMBINER_OPTICS_FAILURE_RATE	Failure rate of combiner optics	0.00417	$months^{-1}$
COLLECTOR_OPTICS_FAILURE_RATE	Failure rate of collector optics	0.00417	$months^{-1}$
COMBINER_BUS_FAILURE_RATE	Failure rate of combiner bus	0.00417	$months^{-1}$
COLLECTOR_BUS_FAILURE_RATE	Failure rate of collector bus	0.00417	$months^{-1}$
DUAL_BUS_FAILURE_RATE	Failure rate of dual functioning bus	0.00417	$months^{-1}$
MISSION_DESIGN_LIFETIME	Mission design lifetime	60	months
TIME_STEP	Time step for discrete A-matrix methods	0.3288	months
NUMBER_OF_DIFFERENCES	Number of pairs of UV points needed to take an image	512	UV points
TIME_PER_CONFIGURATION	Constant scale factor for time needed UV pair and per configuration	4.63×10^{-5}	months
OVERHEAD_TIME_PER_IMAGE	Overhead time per image	2.78×10^{-3}	months
REQUIRED_COMBINING_SPC	Number of spacecraft capable of combining light required for system to function	1	spacecraft
REQUIRED_COLLECTING_SPC	Number of spacecraft capable of collecting light required for system to function	2	spacecraft
REQUIRED_TOTAL_SPC	Number of total spacecraft required for system to function	3	spacecraft

TABLE E.2 List of user-defined inputs for Reliability and Productivity toolbox

Global Variable Name	Description	Default Value	Units
COMBINER_OPTICS_ THEORETICAL_FIRST_ UNIT_COST	Theoretical first unit cost of combining optics	25	$M
COLLECTOR_OPTICS_ THEORETICAL_FIRST_ UNIT_COST	Theoretical first unit cost of collecting optics	15	$M
COMBINER_BUS_ THEORETICAL_FIRST_ UNIT_COST	Theoretical first unit cost of combining spacecraft bus	20	$M
COLLECTOR_BUS_ THEORETICAL_FIRST_ UNIT_COST	Theoretical first unit cost of collecting spacecraft bus	20	$M
DUAL_BUS_ THEORETICAL_FIRST_ UNIT_COST	Theoretical first unit cost of dual functioning spacecraft bus	30	$M
OPERATIONS_COST_ PER_BASELINE	Cost to operate system	0.1	$M/base-line/month
LEARNING_CURVE_ SLOPE	Learning curve slope percentage	95	%
TOTAL_SYSTEM_ BUDGET	Maximum total amount of money to be spent on the system	360	$M
Comparison and Optimization Tools			
ARCHITECTURE_ MATRIX	Matrix of architectures to be evaluated. Order is [dual com col]. Note: for use with *arch_comparison_full_x.m* architectures order is [dual com col Xmo Xlo Xb].	N/A	[- - -] or [- - - $M $M $M]
AVERAGE_VALUE_FOR _CPI	Average, or normal, value for CpI (cost per image)	0.2	$M
AVERAGE_VALUE_FOR _NOI	Average, or normal, value for NoI(number of images)	1250	images
AVERAGE_VALUE_FOR _REL	Average, or normal, value for reliability	0.8	N/A

TABLE E.2 List of user-defined inputs for Reliability and Productivity toolbox

Global Variable Name	Description	Default Value	Units
WEIGHTING_FOR_CPI	Weighting for cost (CpI) in objective function and "score"	0.4	N/A
WEIGHTING_FOR_NOI	Weighting for productivity (NoI) in objective function and "score"	0.3	N/A
WEIGHTING_FOR_REL	Weighting for reliability in objective function and "score"	0.3	N/A
SCALE_FACTOR_FOR_INCREASE	Scale factor for increasing reliability with money spent	25	N/A
BUDGET_FLAG	0 = Use percentage of cost spc. to find total budget 1 = Use a given total budget	0	N/A
PERCENT_OF_BUDGET_FOR_IMPR	Percentage to use if budget flag = 0	20	%
DELTA_GUESS_FOR_SA_MONEY	Initial guess as difference between two neighboring design vectors' reliability (*arch_comparison_w_impr*)	0.01	N/A
SA_MONEY_ITERATIONS	Number of iterations for SA optimization algorithm for only dividing money (*arch_comparison_w_impr*)	500	iterations
INITIAL_DELTA_GUESS_FOR_SA	Initial guess as difference between two neighboring design vectors' reliability (full optimization)	0.1	varies with objective function (N/A)
SA_NUMBER_OF_ITERATIONS	Number of iterations for SA optimization algorithm	1500	iterations
SA_STEPS_DOWN_IN_TEMPERATURE	Total number of steps down in temperature	300	N/A
SA_DUAL_FUNCTIONING_BOUNDS	Bounds and increments on the number of dual functioning spacecraft for SA	[0:1:5]	spacecraft

TABLE E.2 List of user-defined inputs for Reliability and Productivity toolbox

Global Variable Name	Description	Default Value	Units
SA_COMBINING_ BOUNDS	Bounds and increments on the number of combining spacecraft for SA	[0:1:6]	spacecraft
SA_COLLECTING_ BOUNDS	Bounds and increments on the number of collecting spacecraft for SA	[0:1:9]	spacecraft
SA_MONEY_ON_ COMBINING_OPTICS_ BOUNDS	Bounds and increments on the money spent to improve combining optics for SA	[0:5:100]	$M
SA_MONEY_ON_ COLLECTING_OPTICS_ BOUNDS	Bounds and increments on the money spent to improve collecting optics for SA	[0:5:100]	$M
SA_MONEY_ON_ BUS_BOUNDS	Bounds and increments on the money spent to improve the bus for SA	[0:5:100]	$M
NUMBER_OF_ GENERATIONS	Number of generations to be evaluated for GA	60	generations
POPULATION_SIZE	Population size for GA	50	individuals
CROSSOVER_RATE	Crossover (mating) rate for GA	0.6	N/A
MUTATION_RATE	Mutation rate for GA	0.1	N/A
BOUNDS_FOR_GA	Bounds for design variables for GA. Note: variables listed in order [dual com col Xmo Xlo Xb]	[0:5; 0:6; 0:9; 0:100; 0:100; 0:100]	[- - - $M $M $M]
Sensitivity Analysis Tools			
NUMBER_OF_ SENSITIVITY_POINTS	Number of points used to check the sensitivity to each variable	3	points
CHANGE_IN_ SENSITIVITY_POINTS	Change between each point for each variable for sensitivity analysis (given in decimals, not percentages)	0.05	N/A

TABLE E.2 List of user-defined inputs for Reliability and Productivity toolbox

Global Variable Name	Description	Default Value	Units
ONE_SENSITIVITY_POINT	One point to perturb parameters to. Only used in *sensitivity_one_point.m*	0.05	N/A
PARAMETERS_TO_CALCULATE_SENSITIVITY_TO	Vector of which parameters to calculate sensitivity to. Zero if not calculating sensitivity, one if calculating sensitivity	ones(1,23)	N/A