# Multi-Attribute Tradespace Exploration and its Application to Evolutionary Acquisition

By

Jason Edward Derleth

B. A. Liberal Arts
St. John's College, Annapolis 2000

Submitted to the Department of Aeronautics and Astronautics Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

at the
Massachusetts Institute of Technology
May 2003



Signature of Author……
Department of Aeronautics and Astronautics
May 23, 2003

Certified by……
Eric Rebentisch
Research Associate, Center for Technology, Policy, and Industrial Development
Thesis Supervisor

Certified by……
Wesley Harris
Professor, Department of Aeronautics and Astronautics
Aeronautics and Astronautics Reader

Accepted by……...
Edward M. Greitzer
H.N. Slater Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

# Multi-Attribute Tradespace Exploration and its Application to Evolutionary Acquisition

By

Jason Edward Derleth

Submitted to the Department of Aeronautics and Astronautics Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

ABSTRACT:

The Air Force has recently embraced Evolutionary Acquisition (EA) as its acquisition strategy of choice. EA is an especially difficult method of acquisition and presents some extraordinary challenges at the system engineering level.

Multi-Attribute Tradespace Exploration (MATE) is a tool at the system engineer's toolbox that can provide some focus on a project in EA. MATE, a tool initially developed by Adam Ross and Nathan Diller at MIT, is a method of developing models to simulate the product user's preferences for the attributes of a design. Once these preferences are well known, they can be used to guide the design choice.

The design choice is further guided by the creation of system level computer models that represent the design choices available to the engineer. These choices are then varied systematically to create a "tradespace" of possible designs. This tradespace exhaustively enumerates all of the possible design choices for the engineer. Then, through the preference models previously developed, each possible design is ranked in order of user utility and cost. The result can be graphed, giving a visual representation of the utility and cost of literally thousands of architectures in a single glance.

This thesis shows that MATE is a useful tool for a systems engineer working on an EA system. There are many benefits to the use of MATE in EA, including but not limited to: a better understanding of the end user's desires and requirements for the system; the ability to optimize the system for the first evolution; the possibility of understanding what will become optimal in later evolutions; quick redesign time if circumstances or preferences change; and further insight into systems level considerations.

In addition to showing some of the benefits of MATE, this thesis furthers the application of MATE itself into systems not involved with space. Previously all applications of MATE had been concerned with space systems.

## Acknowledgements:

The work presented in this thesis was performed under the direction of the Lean Aerospace Initiative. LAI is a partnership among industry, government, labor, and MIT aimed at bringing about fundamental change in both the space and aircraft industries of the United States. I sincerely appreciate LAI's support and willingness to take a risk on bringing a liberal arts student to the technical world of MIT.

There are several people whom I would like to thank. Without the help of my advisor Joyce Warmkessel, I would not have been able to enjoy the full "MIT experience." Thank you, Joyce. Cancer took you from us before this thesis was finished, and I am very sorry that you won't be here to see me graduate. We were always close, and that helped me a great deal.

Eric Rebentisch is my advisor now, and I would like to thank him for all of his help over the past few months working with me on this paper. He has helped me make it a much stronger thesis, and reined in my enthusiasm when appropriate.

I would like to thank that wonderful group of people that are, to my great benefit, my friends: Stephanie Bahramian, Heidi Davidz, Chris Roberts, Adam Ross, Daniel Sanford—I could never have made it through MIT without you around to listen to my complaints!

Finally, I would like to thank Professor Wesley Harris, who believed (and continues to believe) in me, helped me understand, and had time for me whenever I chose to ask him.

## Executive Summary:

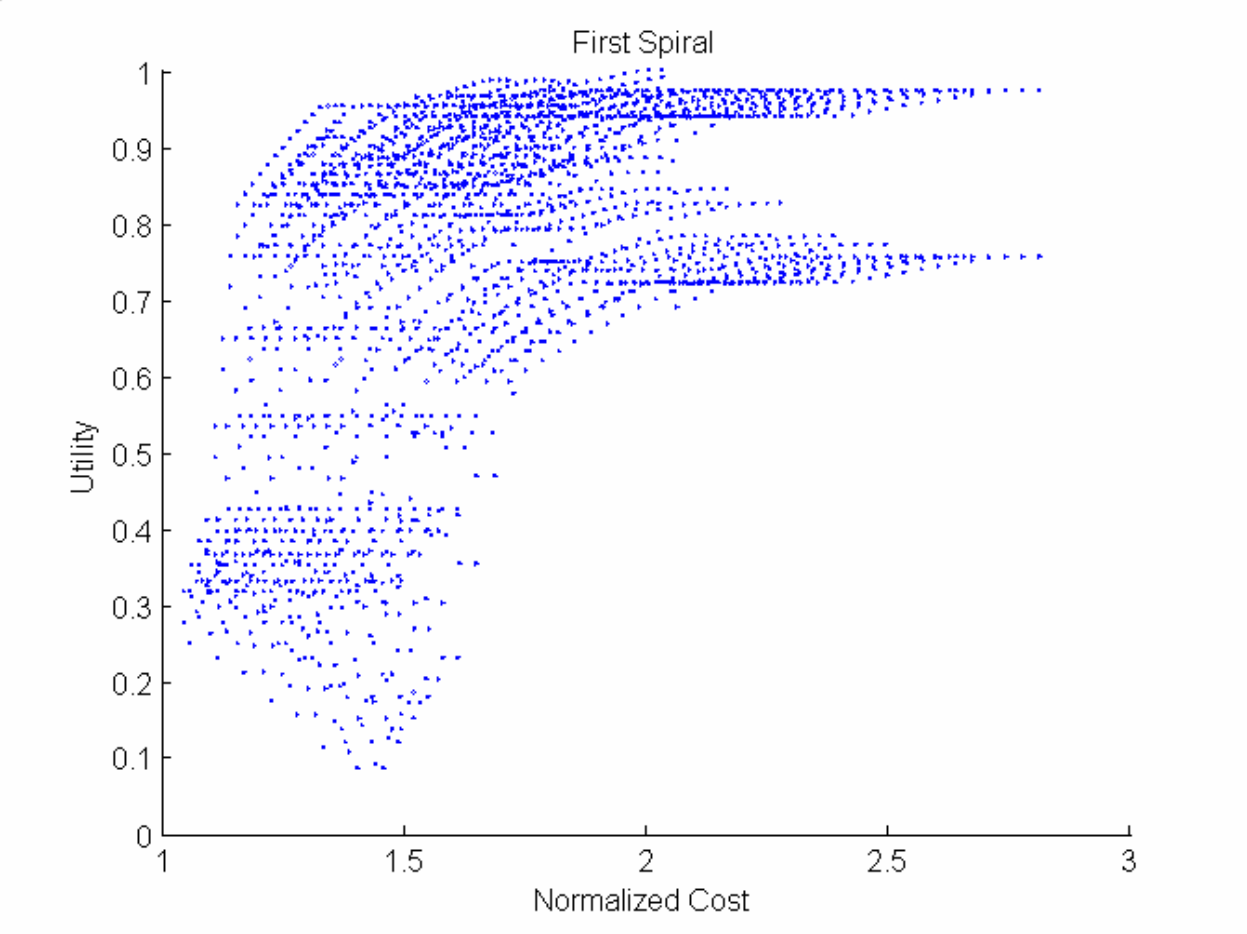
The Air Force has recently embraced Evolutionary Acquisition (EA) as its acquisition strategy of choice. EA is an especially difficult method of acquisition and presents some extraordinary challenges at the system engineering level.

Multi-Attribute Tradespace Exploration (MATE) is a tool at the system engineer's toolbox that can provide some focus on a project in EA. MATE, a tool initially developed by Adam Ross and Nathan Diller at MIT, is a method of developing models to simulate the product user's preferences for the attributes of a design. Once these preferences are well known, they can be used to guide the design choice.

The design choice is further guided by the creation of system level computer models that represent the design choices available to the engineer. These choices are then varied systematically to create a "tradespace" of possible designs. This tradespace exhaustively enumerates all of the possible design choices for the engineer. Then, through the preference models previously developed, each possible design is ranked in order of user utility and cost. The result can be graphed, giving a visual representation of the utility and cost of literally thousands of architectures in a single glance.

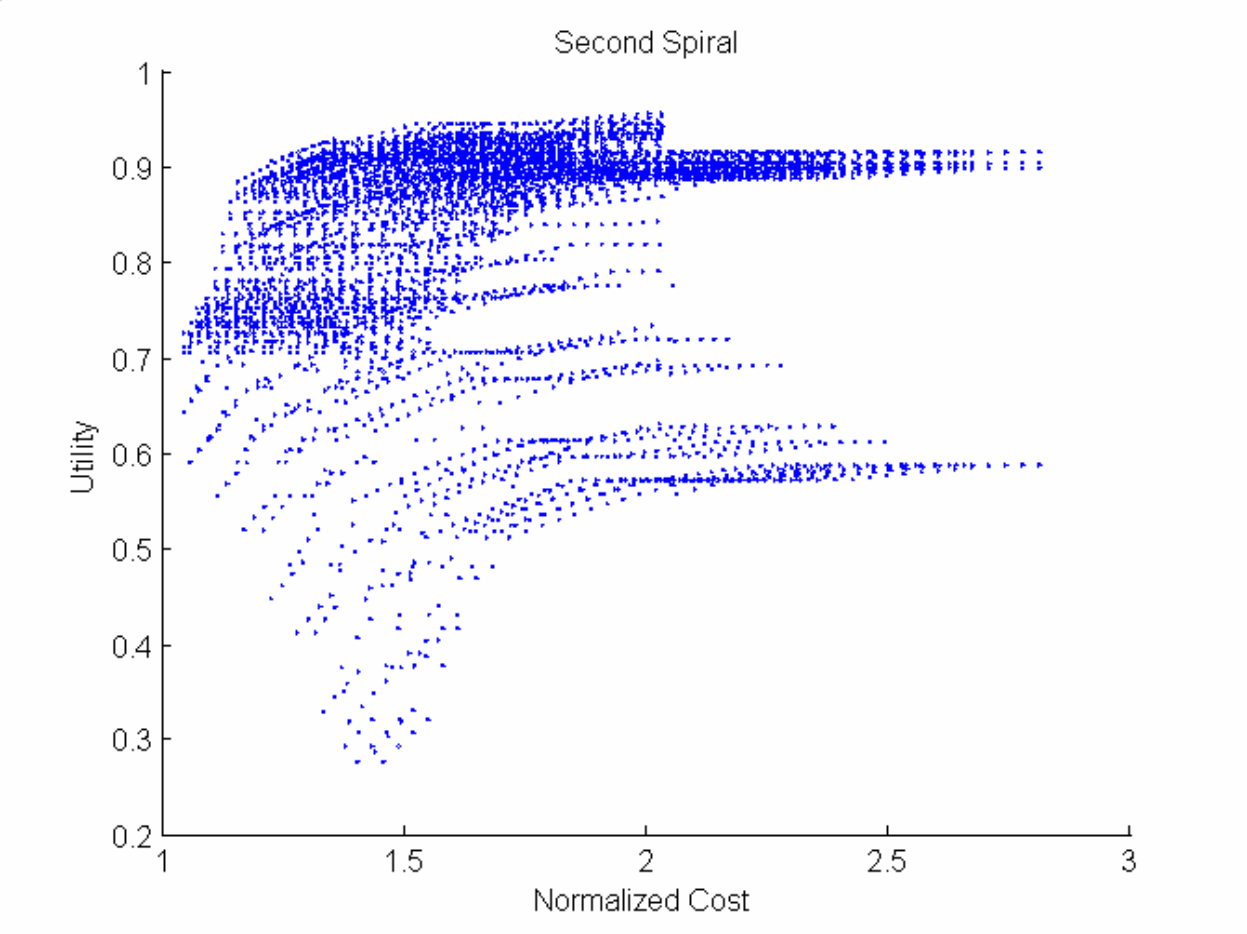
It might be possible to create multiple models for a system that anticipates the different evolutions that might be implemented on it. Through such models, intuition about the initial system configuration might be enhanced, and the initial architecture choice modified based on the information gained from this modeling.

This procedure is developed and implemented throughout this thesis on a sample EA candidate: the Small Diameter Bomb. (SDB) Three different evolutions are built: first, the SDB is designed as a small JDAM with wings, giving it the high accuracy and standoff distance that the Air Force requires. The tradespace for this iteration is below:



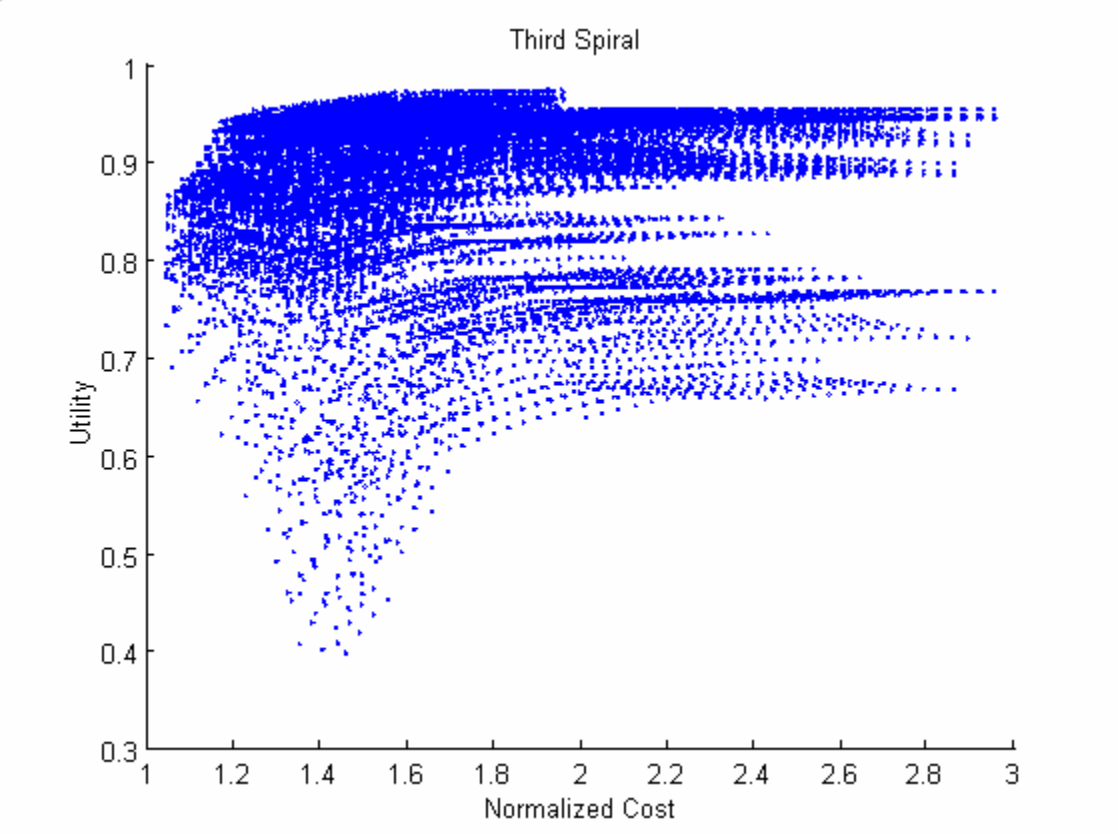

This graph shows that there are thousands of sub optimal architectures, many of which are nearly identical to the optimal ones. They may differ only in the size of the wings, or perhaps in the diameter of the bomb, or even in the amount of explosive carried. It is very difficult to optimize even a simple system, especially when attempting to consider multiple evolutions simultaneously. The pareto-optimal frontier is the set of points along the left and top portion of the graph, and includes all of the architectures that the user must pay a higher cost to have a higher utility, or must sacrifice utility to have a smaller cost.

In the second model, the ability for the bomb to retarget in mid flight is added. Although this evolution requires very little physical change to the bomb, changes are seen in the tradespace:




It becomes relatively less expensive to reach the medium utility architectures, however, it is no longer possible to reach full user utility. The system is simply too technically complex. Additionally, many of the architectures on the pareto-optimal frontier were not on the frontier for the first evolution.

Finally, in the third model, the ability for the bomb to loiter over its target is added to the models:




This architecture includes loiter time, the time in minutes that the bomb can stay over a target area. This allows the Air Force to control an area without endangering lives or expensive fighter jets. As can be seen from the graph, there are now tens of thousands of outclassed architectures, many of which are simply not carrying the correct amount of fuel.

Again, the pareto-optimal frontier changes. To have an architecture on the frontier that started on the frontier in the first evolution is very difficult; only the least expensive of the frontier architectures remains optimal throughout the three evolutions.

An analysis of these three tradespaces is preformed, attribute by attribute, with several interesting results.

The first result of interest is that the architectures that are optimal in the first evolution are not always optimal in the second evolution and rarely optimal in the third evolution. The reason for this is that the user wants different things from the SDB system at different points in time.

Although the attribute of “sorties required to destroy target set” is of lesser importance, it turns out to be of great importance in the overall effectiveness of the architectures.

Another interesting result is that much of the modeling is reusable between evolutions. While building the first model took several months, once that model was constructed, the second and third models were easily added on to the system.

Finally, it is concluded that MATE is a possibly effective tool for the systems engineer who is working within an Evolutionary Acquisition program.

# 1 Introduction

## 1.1 Motivation

### 1.1.1 Evolutionary Acquisition

Modern war machinery is complex. New projects can spend years in the design and test phases of production. In the current global climate, military needs can change drastically in two years, often making these carefully designed and tested machines obsolete before they are ever used. The Air Force has recently adopted 'Evolutionary Acquisition' (EA) as a new type of acquisition strategy that will hopefully produce better systems and put them in the hands of the warfighter more quickly than ever before.

EA is a sea-change for the acquisition process. It works in 'spirals'—a spiral is a quickly produced design, using available technology, and is meant to get new tools into the warfighter's hands as quickly as possible. Ideally, a spiral will have a product in the field 12-36 months after initial concept completion (AF Instruction 63-123, p. 2). After the engineers are done with the design of the first spiral (before testing is complete) they begin designing the next spiral. In this spiral, more information has been gathered during testing, technology has advanced somewhat, and the military situation that the system is meant to address has advanced, thus possibly changing the warfighter's desires. All of this information is used to build on the first spiral to make the second design better. Spirals continue to improve the product until the warfighter has something that satisfies all of his needs.

Preliminary design will be paramount to making EA work. Engineers working in the preliminary design choose the materials, tools, and processes needed to build the system; these,

in turn, determine approximately 80% of the cost of the system. Information from the preliminary design of spiral #1 will need to be built upon and re-used to help guide the preliminary design of spiral #2. Since the system may change dramatically between spirals, the tools involved in preliminary design need to be flexible. Since the design is meant to be accomplished quickly, the tools must be easy to use, accurate, and quick.

Multi-Attribute Tradespace Exploration with Concurrent engineering (MATE-CON) may be a tool that accomplishes everything that must be done to make EA work. MATE-CON consists of two phases, an architectural level study followed by a preliminary design using concurrent engineering.

In the architectural level study, engineers begin with the use of tools developed by social scientists to model the user's or customer's preferences. Through multiple verification iterations of the initial interview process, both the user and the engineers involved in the process gain substantial intuition about the project, allowing the engineers to produce a better design. The user's preferences are then aggregated into a single utility function. This allows the comparison of the utility of different systems. Engineers then build architectural-level parametric models of the system to simulate the system's performance. These parametric models allow the engineers to enumerate a tradespace of architectural designs. By using a parametric cost model, this tradespace can be graphed as a scatter plot on a single x-y set of axes, with one axis as cost and one axis as utility. A restricted, optimal set of concepts can then be considered by the user, and a single design chosen.

The second phase of MATE-CON is concurrent engineering. Higher level tools and models are built and used to produce a preliminary design. The engineers have built up a high level of intuition about the user's preferences and the system itself, and, through the use of further tools

their choices in these design sessions can be guided by the same utility function developed and verified in the first segment. In essence, this process allows the user to gain a technical voice during the preliminary design process that will help guide the design.

## 1.2 Value

The MATE-CON method is useful for anyone attempting rapid design or rapid design iterations. It is also an optimization tool, and will thus be of assistance to anyone who wishes to begin optimization at the architectural level.

Ideal candidates for using MATE-CON are: government agencies and supporting agencies such as FFRDCs, who can use the MATE section of this method to spec out a system before writing requirements; space systems engineers; teams in the Analysis of Alternatives phase of a government contract.

## 1.3 Scope

The scope of the thesis includes the development and assessment of the first phase (architectural level of design) of the MATE-CON process within an EA framework. The architectural level of design is deemed appropriate for proof-of-concept, the benefits of concurrent engineering with MATE having been explored by Diller and Ross in previous papers from MIT (Ross '03, Multi-Attribute Tradespace Exploration with Concurrent Design as a Value-Centric Framework for Space System Architecture and Design, and Diller '02, Utilizing Multiple Attribute Tradespace Exploration with Concurrent Design for Creating Aerospace Systems Requirements.)

This thesis will attempt to explore whether MATE-CON is an appropriate tool for Evolutionary Acquisition by modeling three spirals and tracking several initial designs through

the second and third spirals. It is hoped that some insight can be gained into spiral acquisition itself from this exploration.

The system modeled will be the Small Diameter Bomb (SDB), a 250-lb class munition designed to be as effective as a 1,000 lb class bomb while minimizing collateral damage. This system has the ideal qualities of being simple enough to model within the time constraints of a master's thesis while providing interesting tradespaces and multiple spirals to explore. This will mark the first time that MATE-CON has been applied to an aeronautical system, as all previous iterations of this system have been applied to space systems.

## 1.4 Methodology

First, MATE interviews are conducted. After the MATE interviews are complete, mathematical modeling of the system is preformed. After the modeling is complete, the interviewee verifies the model to be certain that his or her true preferences have been captured.

### 1.4.1 MATE Interviewing Methodology

Since this process has been done several times in an ad hoc manner, it was intended to perform MATE from start to finish in the proper manner. To accomplish this, a search was made for an official user or decision maker to perform the MATE interviews.

An advanced concept design engineer at a defense contractor that submitted a proposal to the SDB program was interviewed. A short presentation was made to this engineer. This meeting was extremely useful in building knowledge about the SDB system, but, unfortunately he was not available for the MATE interviews.

A second interview was preformed with a member of the AF/XORW. Again, this meeting was extremely helpful in building knowledge about the SDB system. This contractor suggested

that what I was doing would have been an excellent tool at the beginning of the Analysis of Alternatives (AOA), but the tool was too high of a level to be of use now.

Strangely, his thorough knowledge of the system was a significant barrier to his performing a MATE interview. He was unable to disconnect performance characteristics from the actual system that he had spent so much time with. It has been seen in previous applications of the MATE process that the user's mode of thinking is the biggest barrier to effective implementation, and that was the case here.

Unfortunately, this marked the end of the search for an official user. A surrogate user was found to take the MATE interviews: a USAF officer with a background in weapons program management agreed to perform the interviews as if he were a pilot who would use the SDB while flying missions.

### 1.4.2 **Mathematical Modeling Methodology**

Matlab is used to perform the modeling. A series of individual models are created, allowing each module to be independently verified. Also, these individual models can be turned on or off easily, new modules can be added on or removed when appropriate. Such a modular software architecture is exceptionally useful for using MATE in Evolutionary Acquisition, as the entire model can be altered radically with only a few keystrokes. Code reuse is made simple; if a module needs to be modified for the new spiral, it can be done quickly. If a module needs to be added, only the module must be written, so long as the data structure is the same.

The modeling was kept as simple as possible without sacrificing too much realism. As the author's knowledge of the system grew, the models changed and grew more complex. Given a team of engineers working on a project full time, the author believes that the modeling would become quite complex and could represent complex systems quite well.

It should be noted that modeling the SDB taught the author about the system. A great deal of insight is gained when one spends time trying to imitate reality with a computer. The side benefit of MATE is that the engineers build intuition about how the architectural choices at hand affect user utility. In a sense, the guidelines set by the user's utility help the engineers understand what is useful and what is not. It becomes easier to meet or exceed the expectations and desires of the user once one thoroughly understands what those expectations and desires are.

# 2 Literature Review—the "State of the Art"

## 2.1 Overview

Since this work is an attempt at modeling the Small Diameter Bomb system, several categories of literature should be reviewed to determine what the state of the art is. This is a review of evolutionary acquisition, Multi-Attribute Tradespace Exploration, existing methods for architecting systems, mathematical methods of modeling systems, and weapon design.

## 2.2 Evolutionary Acquisition

This author began his journey with the beginning of MIT's Lean Aerospace Initiative's (LAI) initiation of a team to work with the Air Force's Acquisition Center of Excellence. (ACE) Mr. Derleth was able to participate in this team, and began to read about the ACE office's preferred method of acquisition: Spiral Development, also known as Evolutionary Acquisition. (EA)

### 2.2.1 Air Force Instruction 63-123

In AF Instruction 63-123, "Evolutionary Acquisition for C2 Systems," 1 April 2000, the Air Force outlines exactly what is meant by Evolutionary Acquisition (EA) for Command and

Control (C2) systems. Within this document lies the rationale for EA itself: Since technology cycles are reducing, requirements evolve through time, a rapid, adaptable acquisition process is needed. Traditional acquisition techniques are slow and inflexible, so the idea of EA came about.

The basic idea of EA is to develop and field a core capability quickly, with a goal of 18 months or less, with the intent to develop and field additional capabilities through time. This document names the design cycles "spirals." Each spiral increases the capabilities and is done quickly.

### 2.2.2 **Air Force Guide to Evolutionary Acquisition.**

While the previous document is specific to C2 systems, the reasons for and the benefits of EA are not. The "Air Force Guide to Evolutionary Acquisition," draft of November 2000 from the SAF/AQ Evolutionary Acquisition Reinvention Team, states that technology cycles have diminished and that the world situation since the end of the Cold War have both contributed to changing the world. In addition to these factors, the Guide also remarks:

> In a fiscally constrained DoD budget, it is likely that new system starts will be few, modifications to current systems will be the norm, and use of nondevelopmental items will be emphasized.

In essence, this is suggesting that, due to the world condition and to budgetary constraints, Evolutionary Acquisition will become the norm regardless of the desires of program managers.

### 2.2.3 **Adaptive Software Development, James A. Highsmith III**

The ACE office's enthusiasm for Evolutionary Acquisition seems logical, but EA originally comes from software development. Time should be spent looking at how well EA works in software, and why. With this information, one can examine whether or not EA is applicable to highly complex physical systems, such as bombs, jets, satellites, or other systems.

EA and spiral development occurs in software development for two main reasons: first, it is easy to make a program that meets a base set of customer's needs and then add more complexity to the program in future spirals; second, software development is based on computer systems, and computer systems have a high rate of increase in capabilities. In essence, new computer systems have a great deal of excess capability compared to their predecessors. Thus, it is logical to make (and customers have the desire for) newer versions of programs to take advantage of this excess capability.

In Highsmith's book, Adaptive Software Development, (Highsmith '99) there is a quick description of the Evolutionary and Spiral development models. He suggests that this is not enough, that it is necessary for teams to be adaptive. In Evolutionary and Spiral development, the team lead still exerts control through setting requirements for each spiral, and driving the team to develop software that meets these requirements. Highsmith suggests that it is better to have no idea where a team is going, therefore extremely profitable side-tracks can be taken advantage of. He titles these side-tracks "breakout ideas," and considers them extremely valuable. (p. 40, Highsmith)

They are valuable because they allow the team to change the goals fluidly towards what they perceive the current customer needs are. They also allow the team to change the coding as good ideas come up. This sort of flexibility is necessary to respond to complex system change throughout time.

It is in part two that Highsmith's writing made a connection in this author's mind: referring to a mission statement's underlying meaning, he states:

> Writing a mission statement is one thing; understanding the scope, meaning, subtleties, ambiguities, and limits of a mission is another thing altogether.

This is an excellent description of one of the crucial aspects of systems architecture. Without someone dedicated from the beginning of a project to attending to these items, they will become obscured and indistinct. It is the systems engineer's job to concentrate on all of these things, but also to help direct the orchestra of engineers, customers, and interactions.

## 2.3 Systems Architecture

### 2.3.1 **The Art of Systems Architecting, by Maier & Rechtin, 2000**

This excellent book begins with a description of how engineering and system architecting are different. It should be noted that systems engineering and systems architecture are two different but similar things. A systems architect is more concerned with the shape of the system at its inception and its utility to the customer or user; a systems engineer is more concerned with making sure that the system integrates into that shape. Thus, a systems architect is one who uses his or her experience and tools to design the overall mission, how it integrates with the exterior interfaces of other systems, and how it fits into society; a systems engineer is one who uses his or her experience and tools to design the mission parts and how the subsystems integrate together to form the whole. Architecting begins with the system's purpose. (p. 10)

Systems architects are more concerned with the system than the particular, and their tools and methods are different. The authors of this book are concerned with the system's interfaces. In fact, they consider these the architect's greatest concerns and the source of greatest leverage to help design a system that is significantly better than preceding architectures. (p. 9)

One of the main categories of tools available to the systems engineer is heuristics. The word comes from the Greek, originating in the word `ευρισκειν, meaning "to find a way" or "to guide." (p. 26) Architecting through heuristics is a way of using rules built through experience to

guide the system. These heuristic tools can be used to help architect a number of systems. To start the reader on his journey through architecture and heuristic tools, this book has chapters covering manufacturing, social systems, and software and IT systems.

However, a main section of this book is concentrated on models. These models take many shapes, from scale models to mathematical systems theory. One of the key features of modeling is the ease of building a model as opposed to building the real system. Its comparative cost is small, but the information gained can be immense. It is important to note that there is no one generalized modeling strategy that will work for all cases; each model will be effective only in its area of interest. (p. 197)

A systems model can be quite useful for the systems architect; several methods have been developed in different fields. Some of these models are physical models of the system, some are 3-d computer models, some are mathematical models of system behavior. Unfortunately, use of these modeling techniques is limited. (p. 219) Modeling is one of the tools available to systems architects, but other tools are just as important. It is in managing the interfaces of the architecture and communication between all interested parties that the art of the systems architect must be applied.

### 2.3.2 **Spacecraft Systems Engineering, ed. by Fortescue, Stark, and Swinerd, 2003**

At first, it seems inappropriate to have a book about space systems engineering in a thesis that explores the Small Diameter Bomb system. However, systems engineering is absolutely important for Evolutionary Acquisition. A systems engineer is a person who communicates between the parties who are providing the product (the engineers) and the parties who have ordered the product (the Air Force, in this case). He or she makes sure that the engineers are providing a system that does what the Air Force wants it to do. He or she also is a go-between

for the engineers, making sure that the entire system works as a unit even though it is likely designed in parts (subsystems).

In EA, the number of engineering cycles is increased. The requirements increase with each cycle. Communication between the parties involved is necessary for EA to succeed. A good systems engineer is absolutely necessary in a situation such as this one.

Systems engineering began with space, in fact, the concept was first used by NASA. Some of the best books on systems engineering are space based, such as Space Mission Analysis and Design (published by Microcosm 1999), and NASA's own NASA Systems Engineering Handbook, SP-6105, published in June 1995. One can extract the systems perspective out of books such as Spacecraft Systems Engineering and apply these concepts to any complex system.

These concepts often begin with Mission Requirements. Requirements are top level requirements that the user of the system places on the system performance. The system is divided into a functional arrangement; with a spacecraft, the subsystems are propulsion, guidance, attitude determination, avionics, and the like. With a bomb, it is likely that these divisions would be along the lines of aerodynamics, weapon delivery system (design of the rack mounting system within the plane), guidance, and payload (the payload must be shaped to have the best effect possible for the task at hand, the materials chosen must be compatible, the triggering mechanism must be reliable). The mission requirements are then split among these various functional lines, allowing each subsystem team to design its own part of the system in such a way that the whole system meets all requirements.

Another division within the system that belongs to the systems engineer is one of time. In space systems, the time from the initial conception of a complex mission to its launch and operation can easily extend over a period of 10 years. This can be much longer if the system is

more successful; Voyager 1 was launched in September of 1977 and is still sending data to us today, so this particular space system has been in operation for 26 years—this does not include its design and manufacture time. It is unlikely that a system such as the SDB will be in continuous use for 26 years, but the entire program should be considered from the beginning, including feasibility, detailed definition, development, manufacture, test, deployment, maintenance, and decommission/disposal. By planning ahead from the beginning, each phase can be transitioned to more efficiently and smoothly, and each phase can take less time.

This particular view presented in Spacecraft Systems Engineering points out one of the main problems with Evolutionary Acquisition: with each new spiral or evolution, the system changes somewhat. Only by planning ahead can deployment, maintenance, and disposal be a smooth procedure. Field mechanics having to service three different kinds of SDBs may tear their hair out in frustration unless engineers think about maintenance while they design the system.

How, then can a systems engineer help guide such complex projects through multiple iterations while still managing system scope, meaning, subtleties, ambiguities, limits, engineers, customers, and interfaces? There are several possibilities, including modeling of the system. Whether the models be mathematical, physical, or computerized, building a model helps the systems engineer build intuition into the system that he or she is modeling.

## 2.4 Mathematical Modeling

### 2.4.1 **Systems Planning and Design, edited by de Neufville and Marks, 1973**

Through more than a dozen case studies, this book covers three major areas of mathematical models of systems: modeling, optimization, and evaluation.

Modeling begins with three system elements that a systems analyst might be called upon to model: production, supply, and demand. Production functions are modeled as efficient processes, with a minimum of wasted work. It is up to the actual project manager to ensure this level of work. Supply functions describe a relatively narrow part of the entire system, but an important one. A model of supplies can help solve problems before they arise by ensuring that production can continue at a normal, efficient process. Demand functions represent the demand for the product by the public.

This section begins with some of the most basic analysis functions and concepts possible and uses these as building blocks to show how the models for the case studies were built. Reasoning is explained and numerous equations are presented to help the reader understand exactly what a mathematical model of a system is and what it should do. The most important part of this section, however, is the concerted effort made on how to apply the principles being taught.

The optimization section deals with several different optimization methods, including both linearized and nonlinear methods. The authors are very careful to show the limitations and the accuracy of each method presented. Optimization of the production models can help a company produce the correct amount of its product; optimization of the supply models can help an organization reduce its cost of goods sold.

The final section, Evaluation, attacks such diverse topics as evaluation of public works to evaluation of decisions about technological systems with social consequences. To do so, five major classes of decision making and evaluation procedures are presented, in order of increasing complexity:

- Discount rate
- Nonlinear value

- Utility Function
- Multiattribute
- Welfare economics

These different classes of evaluation techniques are exemplified through various case studies. This book provides an excellent first introduction to Multiattribute analysis. The authors point out that “decision analysis . . . is a powerful extension of the traditional procedures insofar as it explicitly and systematically accounts for risk.” (p. 297)

It is limited, however: it is inappropriate for use when individuals or organizations are in conflict. (p. 297) In this situation, Multiobjective Evaluation and Negotiation should be used, a specific instance of the decision making class of “welfare economics.” In Multiobjective Evaluation and Negotiation, multiple decision makers and stakeholders are considered; however, there are no established analytical methods for estimating preferences, nor how any differences might be resolved. Thus, while Multiobjective Evaluation is a better tool, it is not one that is readily applicable to systems engineering

It is this book, combined with experience in using Multi Attribute Tradespace Exploration, that led the author of this thesis to the concept of using MATE to model the Small Diameter bomb through several Evolutions. It seemed possible that MATE could be used to:

1) Help the systems engineer gain insight into the system he or she is modeling;
2) Track architectures through different Evolutions to determine if an optimal choice in the first Evolution remained an optimal choice in later Evolutions;
3) Generally determine if MATE is useful in EA.

To begin this exploration, there follows a review of some existing works on MATE and its underpinnings, Multi-Attribute Utility Analysis.

## **2.5** Multi-Attribute Tradespace Exploration and Concurrent Design

The seminal work in this Multi-Attribute Tradespace Exploration (MATE) and Multi-Attribute Tradespace Exploration with Concurrent Design (MATE-CON), was completed during 2002 and 2003 at MIT by Nathan Diller and Adam Ross. The primary resources for this procedure are found in Ross '03. Based on work by Keeney & Raiffa in 1973, first outlined in their book Decisions with Multiple Objectives, MATE and MATE-CON have taken this basic theory to the next level. First, though, we should look more closely at the origins of MATE and MATE-CON.

### 2.5.1 **Decisions With Multiple Objectives, by Keeney & Raiffa, 1993**

In Keeney and Raiffa, the procedure of multi-attribute utility analysis is explained. This procedure's purpose is to assist the decision making of a systems engineer or systems architect in building a design for the system at hand. It does this through applied social science, using the rules determined by social scientists to develop a framework for interviewing a person to reveal their true preferences.

For example, a systems architect might be attempting to help a fire department reduce its response times. This systems architect would interview the fire chief to determine the various important systems components, called attributes, such as: when the first engine arrives, when the first ladder could be put up, when the other engines arrive, etc. (This example is modified from Keneey & Raiffa, ch. 7.)

After decomposing the system into attributes, the decision maker is then interviewed to determine his preferences on specific values of these attributes. In our simple example, the fire chief would be asked questions to help determine, for example, how much more he values a first engine response time of five minutes over ten minutes.

After these results are obtained, a final interview is given to determine relative weights for the overall utility determination. Once the weights have been determined, a product function is used to determine overall utility. Without delving too deeply into the math here, this product function is:

$$KU(x)+1=\prod_{i=1}^{n}[Kk_iU_i(x_i)+1]$$

where $k_i$ are the attribute relative weightings, $u_i(x_i)$ are the individual utility function, K is the normalization factor so that the values all fall between zero and one, and U(x) is the overall utility of the architecture under consideration. (Keeney & Raiffa, P 288)

This utility function is used to determine the utility of a particular architecture under consideration, i.e., specific values are given for each attribute and the utility is calculated. These specific values are given through conceptual modeling of the system. Decisions are aided by knowing the quantitative utility of each architecture considered. It is important to note that the utility scale is simply an ordered metric scale; it is not known whether or not a utility of 0.5 is twice as good as a utility of 0.25; it is simply known that a higher utility is more valued by the interviewee than a lower utility.

Now that the seminal work for utility analysis has been discussed, it is time to bring the two main sources for MATE into play: Adam Ross's master's thesis of 2003 and Nathan Diller's master's thesis of 2002:

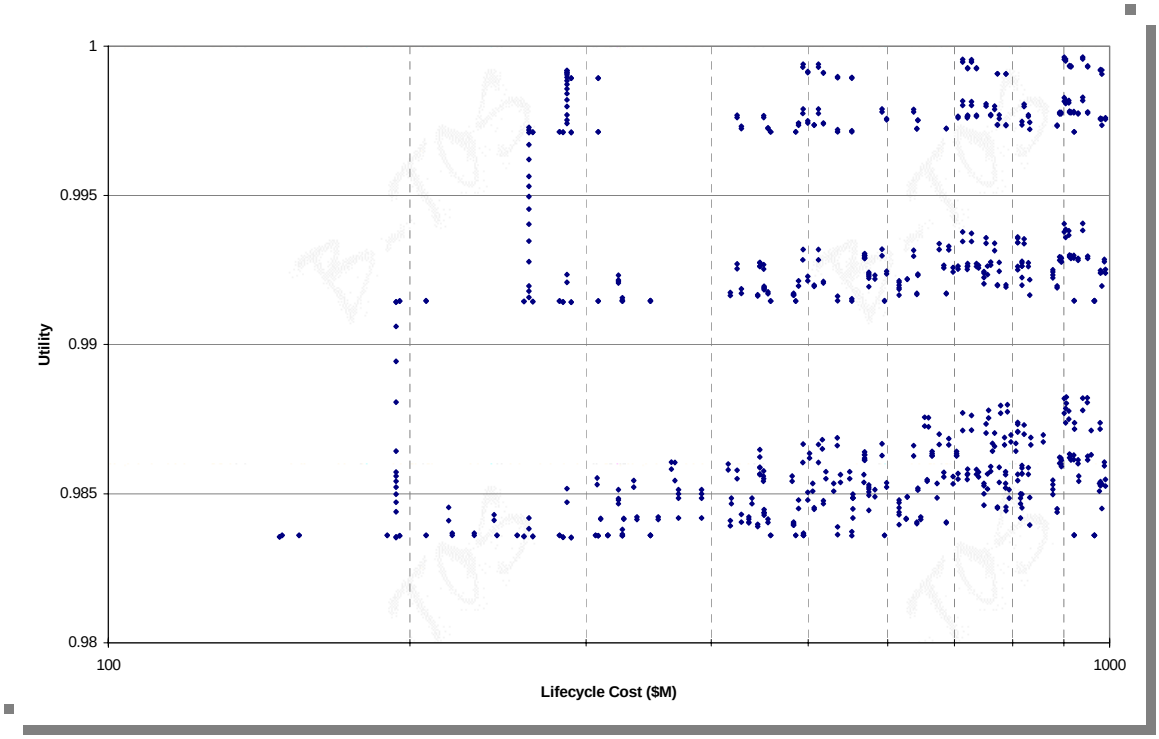
### 2.5.2 **Multi-Attribute Tradespace Exploration With Concurrent Design as a Value-Centric Framework for Space System Architecture and Design, by Adam Ross, 2003**

Ross takes Keeney & Raiffa's ideas and applies them to space systems engineering & architecting. Through the use of modern computing, mathematical models can be built and used

to create a tradespace through the following steps. For a more detailed description of this process, please see Appendix A of Ross, 2003. This excellent appendix is a thorough description of how to implement MATE-CON.

First, a tradespace is developed. In this tradespace are vectors that completely define the system. These vectors not only completely define the system, but they also are extensively varied and combined. Each unique combination of design vectors is a system architecture. These vectors, if the system were a satellite, might include such things as: battery size and number, orbit altitude, inclination, and eccentricity, solar cell size and number, etc. For a bomb system, these design vectors might include explosive weight, guidance type, bomb winglet size, etc. The flexibility of this system is immense; it is limited only by what is possible for an engineer to model.

Since each architecture is a unique combination of the design vectors, and since computers are relatively powerful, it is possible to feed every possible architectural possibility through the utility equations and determine the overall utility of each architecture. Using high level cost models, it is also possible to assign a cost to each architecture. It is then possible to graph these results in numerous ways, thus helping the engineer/systems architect to determine the architectures with the lowest cost and highest utility. The following chart is the lifecycle cost vs. utility chart from the B-TOS development project described in Ross ’03:




Graph from Ross ‘03

It is also possible to determine which architectures are best possible uses of resources: there is a set of architectures for which you must either pay more money to gain more utility, or sacrifice utility to reduce cost. These architectures are collectively called the pareto-optimal frontier. In the above graph, there is an architecture at each ‘knee’ in the tradespace that is the best possible utility for that particular cost.

### 2.5.3 **Utilizing Multiple Attribute Tradespace Exploration with Concurrent Design For Creating Aerospace Systems Requirements, by Nathan Diller, 2002**

Diller ’02 tackles the practical use of MATE-CON within concurrent preliminary design phases through the creation of a computer program that allows the more detailed preliminary design to be analyzed into utility and cost and displayed on the original tradespace, in near real time during concurrent design.

Diller also attempts to incorporate multiple stakeholder utilities—one of the problems raised by de Neufville in Systems Planning and Design. Diller first carefully describes the difference between a ‘user’ and the ‘decision maker’—the decision maker is the person who controls the funding for the project, a user is a person who uses the product. For example, a home computer might be bought by an engineer for use at home. She is the decision maker. Her young son, who will use the computer to play video games, is a user of the system. They have completely different utility functions for the same system.

Another example: a program manager might allocate funds for a new satellite to travel to Mars. He or she is the decision maker, the person who decides to build or to not build the system. There may be several dozen scientists who will all use the data from the system, and each one may have his or her own experiment to put onto the satellite. These users all have different utility from different types of data; the decision maker must incorporate such diversity of desire into his or her decision making procedure in addition to preferences about cost and schedule.

Diller incorporates a “nested utility function” by making the utility of the user one of the attributes in the decision maker’s utility function. Thus, the user’s utility of the current architecture is calculated, and then this number is fed into the decision maker’s utility as a single attribute alongside cost, schedule, and other attributes.

#### 2.5.3.1 Limitations on MATE

Ross and Diller’s theses raise several questions, however good they are: all of the uses within these theses are space systems. Can MATE be applied outside of space systems? Space systems are highly defined systems with a great deal of research and captured knowledge. There are parametric mathematical models for most parts of a spacecraft, and MATE relies heavily upon these models. Can a system like a bomb be parametrically modeled?

Another of the problems with MATE is its newness, its limited application. It is untested outside of the classroom. Despite its success there, it remains to be seen how it can be incorporated into the acquisitions process.

In short, can MATE be utilized outside of a space systems architecting application?

## 2.6 Research Objectives/Questions

Evolutionary acquisition is a difficult procedure. Due to its increased communication, complexity, and speed, a good systems engineer is essential for project success. But what tools are there for a systems engineer to use? Of course, optimization at the architectural or preliminary design level is exceptionally useful, but few tools help optimize for the user's needs.

MATE-CON does exactly this. Through its user or customer focus, through its enumeration, and through its guiding forces throughout concurrent preliminary design, this tool acts as a surrogate user with technical expertise who can participate in the design sessions. In add-on, it fosters communication between user and engineer. Finally, it forces both the engineers and the user to a deeper understanding of the user's preferences. All of these things combine synergistically to allow the engineers to have a design philosophy of "beyond expectations" instead of "meet the requirements."

The guiding questions for this thesis are:

1) Can Multi-Attribute Tradespace Exploration be used in a non-space application?
2) Can MATE be useful for Evolutionary Acquisition?
3) Does MATE help in gaining insight into the system being modeled?
4) Can MATE be used to track architectures through different Evolutions?

The rest of this paper is a presentation of the author's research into the use of the MATE techniques for an EA system, the Small Diameter Bomb. MATE's effectiveness will be explored as thoroughly as possible, and results and conclusions given.

# 3 Multi-Attribute Tradespace Analysis

This section describes the MATE process and the planned development of three small diameter bomb spirals.

## 3.1 Description of Multi-Attribute Tradespace Analysis (MATE) Process

### 3.1.1 MATE Attribute Generation

MATE begins by defining the user preferences. The systems engineer sits down with the user and discusses the system, trying to define a set of attributes that completely defines the system. These attributes should be complete, operational, decomposable, nonredundant, and minimal. (Keeney & Raiffa, p.50)

Completeness: a set of attributes is complete if you can determine how well the system meets its overall objectives by naming values for its attributes.

- Operational: The attributes must be meaningful to the decision maker, and helpful in decision making.
- Decomposable: These attributes should be able to be broken down into parts of smaller dimensionality
- Nonredundant: The attributes should not duplicate any parts of the system.
- Minimal: there should be as few attributes as possible, for complexity of both interviewing and the analysis increases geometrically with the number of attributes.

### 3.1.2 MATE Interviews

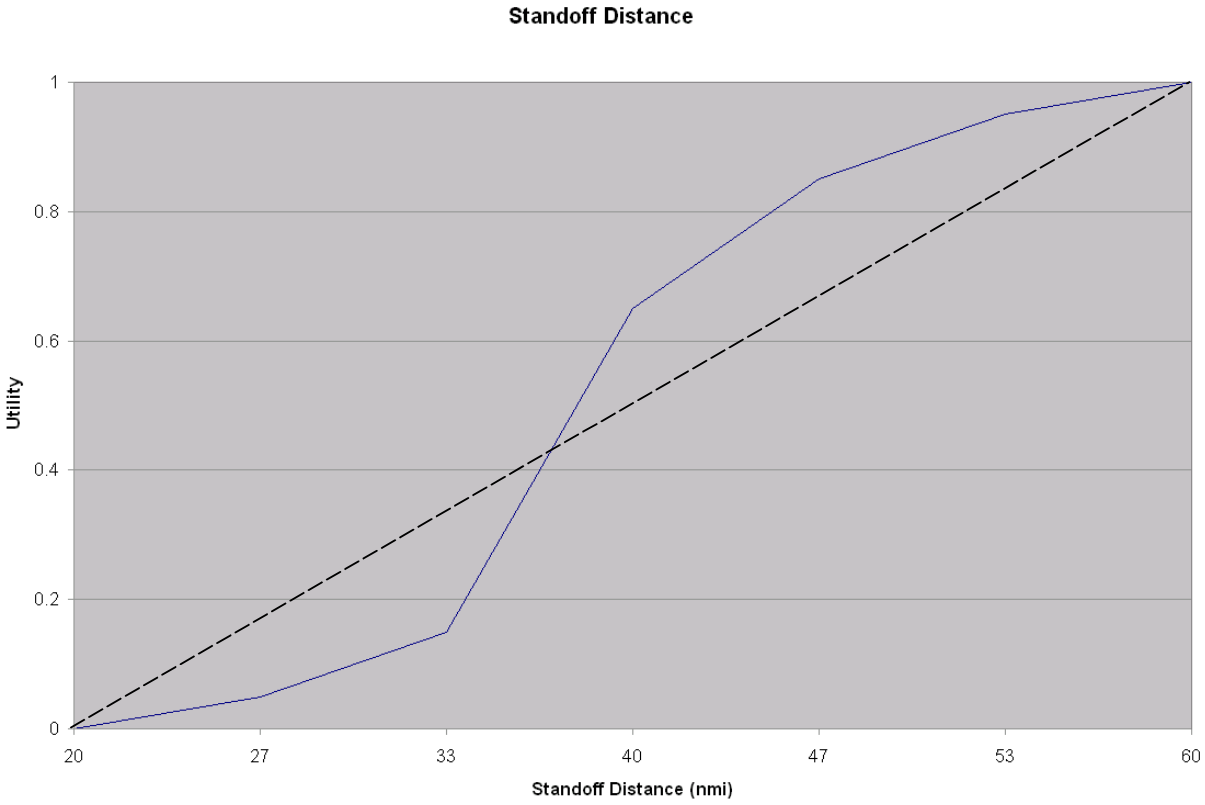
Once the terminology has been introduced and the system analyzed into attributes, the single attribute utility curves must be generated. This is done through user interviews, using the Lottery Equivalent Probability method. This method is the most rigorous way to capture user

preferences. (Ross ’03) The interview process is quite foreign to most people, though, and it may take a few interviews before the user catches on. For a thorough understanding of the social science that has gone into determining that this is the best form of interview to capture user preferences, please see chapters 4, 5, and 6 of Keeney & Raiffa.

An example of a question within this interviewing process is:

> Scenario: New wings have been developed for the Small Diameter Bomb. They will possibly increase the Standoff Distance of the SDB. Engineers indicate that, due to uncertainties in design, the current wings will have a 50-50 chance of getting a Standoff Distance of 60 nautical miles or of 30 nautical miles. The new wings will give a 30% chance of getting a Standoff Distance of 60 nautical miles or a 70% chance of getting a Standoff Distance of 20 nautical miles. Which would you prefer, the current wing design, or the new wing design, or are you indifferent between the two?

This question would be asked several times, varying the percentage of success with the new wings. After several iterations of this question, varying the percentages each time, an indifference point would be achieved. The ranges of the current architecture are then changed, for instance to 20 and 60 nmi, and the questions are repeated until indifference is achieved. Through several iterations of these question cycles, the user’s single attribute utility preferences are determined. An example of a utility curve is shown here:




These curves show risk preferences for the user for a single attribute. A risk neutral dashed line has been superimposed on this graph. Where the line is above the risk neutral line, then the user is risk averse; where the line is below the risk neutral line, the user is risk prone. The curve above shows that the user is willing to take chances to gain a small amount of standoff distance. However, once a certain amount of distance has been guaranteed by the system, he or she is less willing to invest in risk to gain more standoff distance.

The most expedient method for these interviews is the MIST software, a program that runs in Microsoft Excel, developed by Satwik Seshesai in 2002 at MIT. This allows the user to take an interview with no time constraints. In addition, the program allows a single attribute interview to be taken, then the data can be saved and the other interviews taken later.

### 3.1.3 **Modeling and Verification**

Once the interviews are completed, then the engineers model the system (more information on modeling will be presented throughout this section), usually evaluating thousands of architectures. After the modeling is complete, the results are verified with the user. As the interview procedure is a new one, the interviewing process is unfamiliar to many people. It has been this author's experience that interviewees, once they are familiar with this process, are able to give accurate interviews quite quickly. However, since it is a new procedure, the user's preferences are not always captured on the first try. If this is the case, another interview is taken, and that data inserted into the models. The models are reusable, and new data can be generated in a matter of minutes do days, depending on the complexity of the system.

## 3.2 Evolutionary Acquisition with the Small Diameter Bomb

### 3.2.1 **Overview**

To show whether or not the MATE procedure is worthwhile for a system slated for Evolutionary Acquisition, a specific system is modeled through three Evolutions, or "Spirals." (The difference between EA and Spiral Development is a slim but important one, but one that is ignored for the purposes of this paper. For a better description, see Roberts 2003 or Ferdowsi 2003) and each individual possible architecture is tracked through these spirals to see if any insight can be gained on the system through the use of this procedure.

To determine in advance the spirals that a system is likely to take is, in a way, completely counter to the idea of EA. One of the distinguishing factors between Pre-Planned Product Improvement (P3I) and EA is that in P3I one plans the future development of the product from the start, whereas in EA one allows the system to morph and change as is needed. However, as demonstrated by MIT's X-TOS project, completed as a part of the 16.89 Space Systems

Engineering class in spring of 2002, the MATE procedure allows changes in a complex system to be incorporated quite literally overnight even in the preliminary design phase. It is assumed that such flexibility will allow the user of the MATE system within Evolutionary Architecture to plan ahead without sacrificing any flexibility to respond to a changing world.

### 3.2.2 **SDB Spirals Considered with the MATE Process**

#### 3.2.2.1 Spiral One—A small JDAM with wings

The Small Diameter Bomb's first spiral is a fairly simple system; in essence, the bomb at this point is meant to be a small Joint Direct Attack Munition with wings added to it. Its purpose is to be a small weapon that will fit into the bomb bays of the F/A-22 and, through exceptional accuracy, be as useful in destroying targets as a 1,000 pound bomb. In addition, it will be a large standoff distance weapon, i.e., it will be effective even when launched dozens of miles from its target, keeping the plane out of harm. The accuracy is brought to the bomb by a guidance unit and a steerable tailfin kit, where actuators move tailfins to steer the bomb to its target; the standoff distance is brought to the bomb by supersonic launch and extendable or unfoldable wings. The user's desires are fairly well known on this spiral, as it is near design completion.

#### 3.2.2.2 Spiral Two—Retargeting Capability

The SDB's second spiral may include some autonomy. In this thesis, it is assumed that some autonomy is added, taking the form of retargeting and/or tracking ability. This is a very interesting spiral, showing the flexibility of the MATE approach, for the general configuration of the bomb is not changed at all. Rather, the cost is increased by a delta to account for the cost of programming the system, and a new attribute is added to determine how effective the retargeting is. As it is also possible to add tracking capability at this juncture, the effects of that highly complex system are added without details on how it would be accomplished.

#### 3.2.2.3 Spiral Three—Loiter Time

To make the bomb even more effective, the ability to loiter over an area of interest is assumed to be the purpose of the third spiral. This adds a fuel tank and a small turbojet engine to the system in order to allow the bomb to loiter in this way.

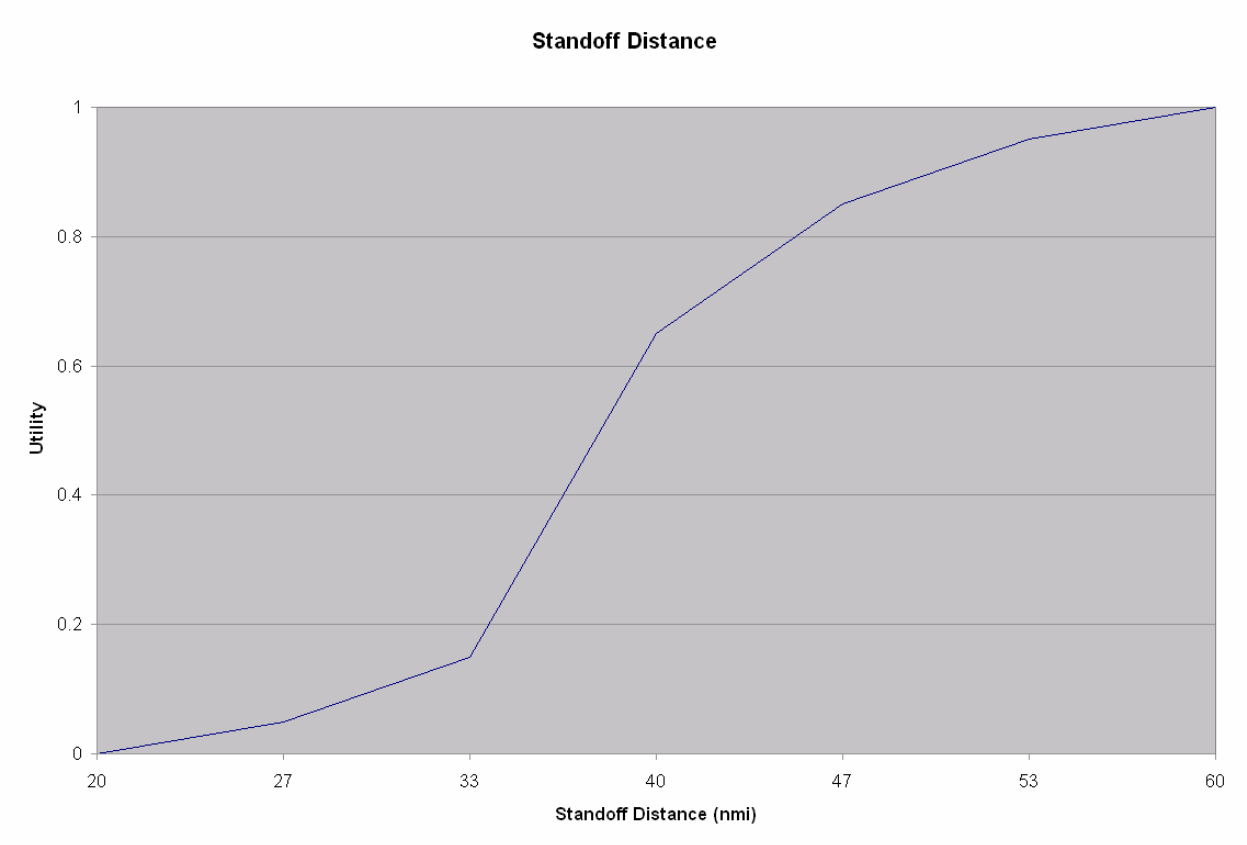
## 3.3 Description of MATE Attributes for each Spiral

### 3.3.1 **Standoff Distance**

Standoff distance is the maximum distance from the target that the bomb can be released. The minimums and maximums for this attribute were given by the surrogate user/decision maker and were in good accordance to information publicly available. The interview scenario was as follows:

> Scenario: New wings have been developed for the Small Diameter Bomb. They will possibly increase the Standoff Distance of the SDB. Engineers indicate that, due to uncertainties in design, the current wings will have a 50-50 chance of getting a Standoff Distance of 60 nautical miles or of 20 nautical miles. The new wings will give a # chance of getting a Standoff Distance of 60 nautical miles or a 1-# chance of getting a Standoff Distance of 20 nautical miles.

The interview was preformed with the MIST software, developed by Satwik Seshesai at MIT. The resulting preference curve is presented here:



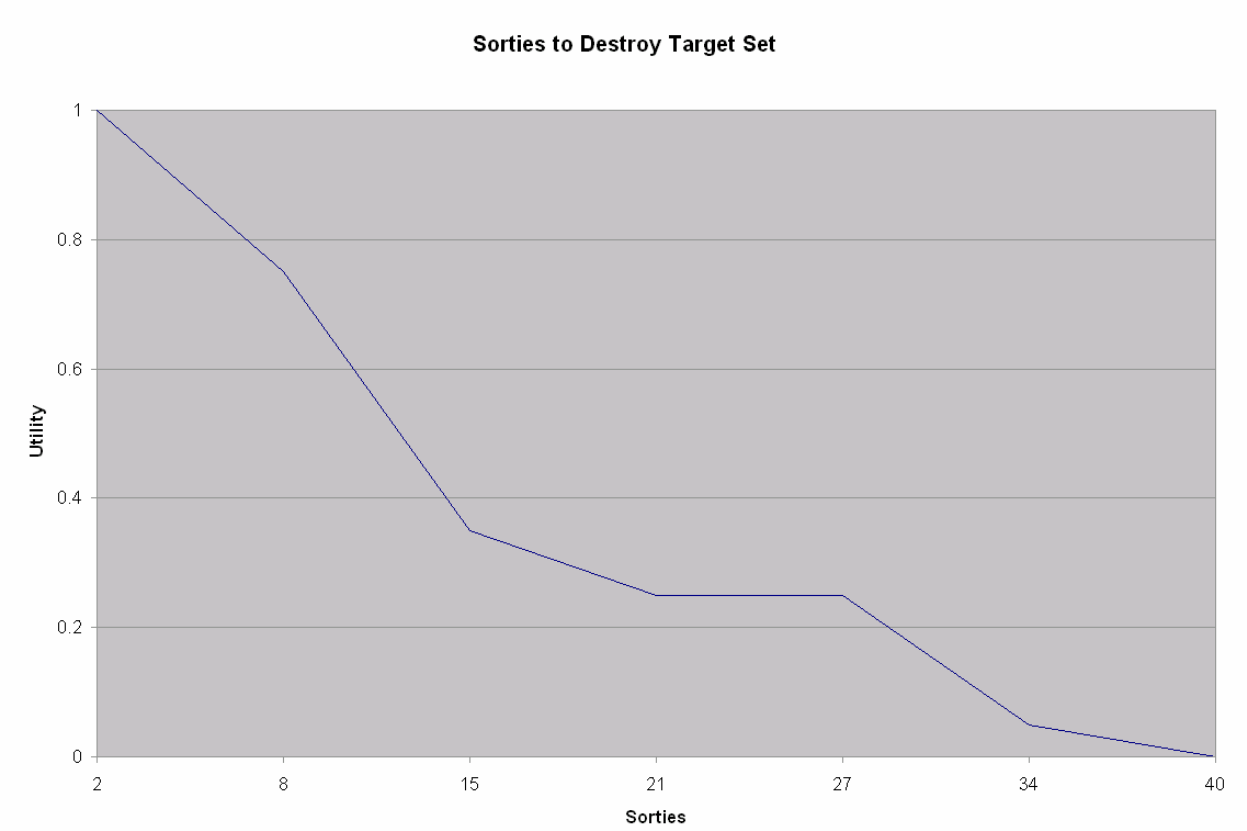

### 3.3.2 **Sorties Required to Destroy Target Set**

Sorties required to destroy target set is the number of planes that must be flown to destroy the chosen target set, assuming that each plane carries a full load of SDBs and no other weapons. It is important to note that a sortie, in this case, refers to a single plane. If three planes fly the mission together, then this would be considered three sorties. The target set was set at five bunkers and several targets of opportunity, such as tanks, armored personnel vehicles, and jeeps. The minimums and maximums for this attribute were determined in discussion with the surrogate user, with the maximum utility set at 2 sorties because, according to him, the Air Force would not send a lone jet to accomplish a mission. The MIST scenario is duplicated here:

> Scenario: A new bomb configuration is being tried. The old configuration has a 50-50 chance of destroying the target set in

> either 2 sorties, or, due to possible problems inherent in guidance, and targeting, in 40 sorties. The newer configuration has a # chance of destroying the target set in 2 sorties and a 1-# chance of destroying the target set in 40 sorties.

The resulting Preference curve was found:



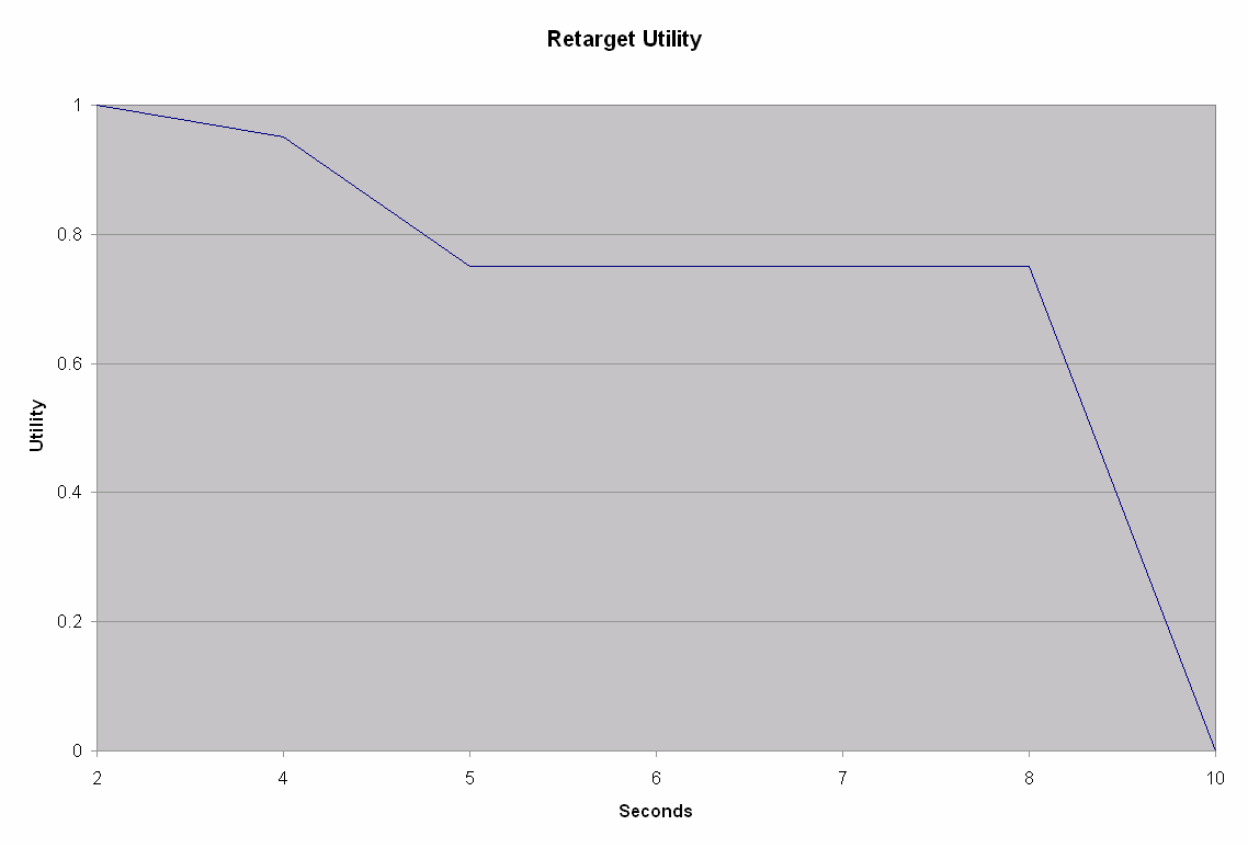

### 3.3.3 **Retarget Time—used in second spiral**

Retarget time is the time in seconds before impact that the bomb can be retargeted and still maintain the same accuracy. The target is assumed to be twenty degrees from the current flight path. The minimums and maximums were determined by the physical model and were set at 0 seconds and 10 seconds. The target is assumed to be swerving at the time of retarget, adding an additional small error. The interview scenario is as follows:

> Scenario: A new targeting system has been designed that will allow retarget. Due to uncertainties in flight, the old system has a

50-50 chance of maintaining the same accuracy when retargeted 5 seconds before impact or maintaining the same accuracy when retargeted 10 seconds before impact. The new system has a # chance of maintaining the same accuracy when retargeted 5 seconds before impact and a 1-# chance of maintaining the same accuracy when retargeted 10 seconds before impact.

The Preference curve is presented here:

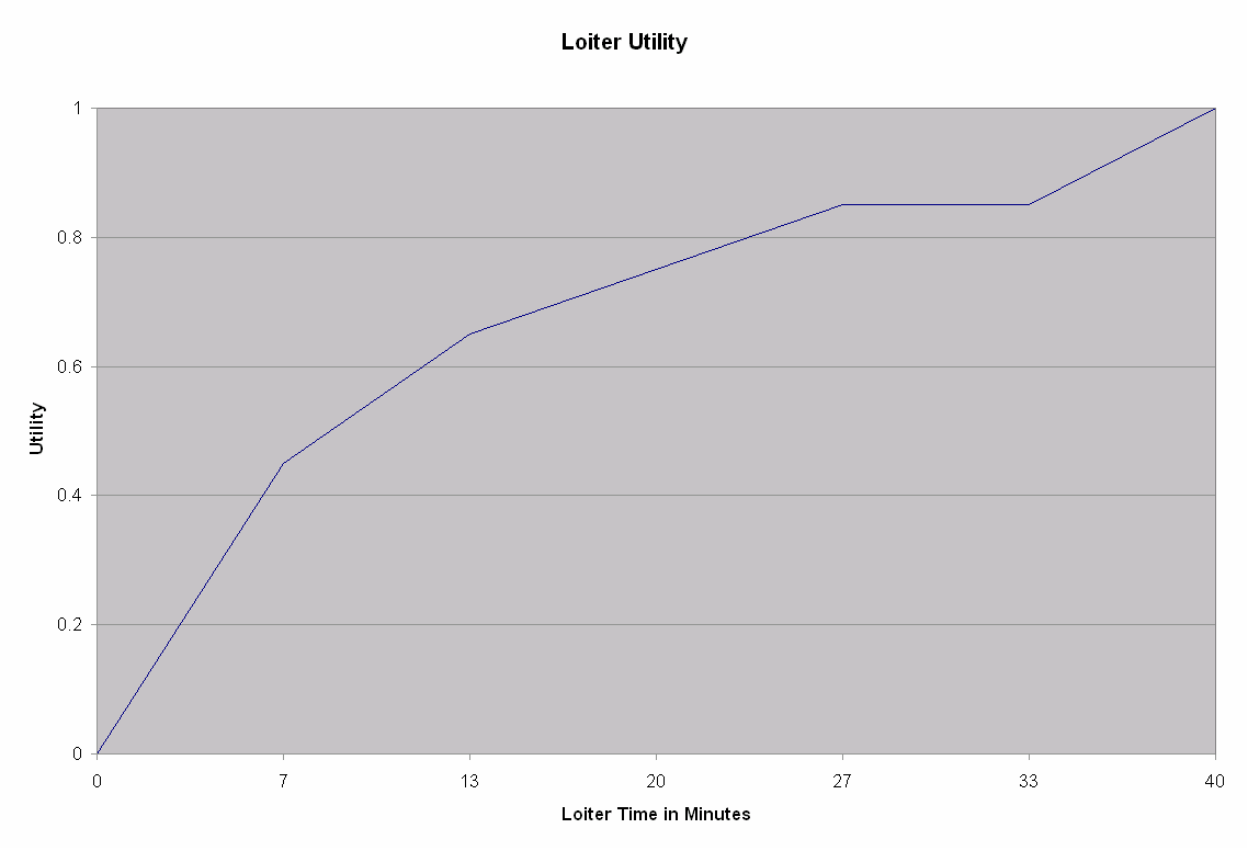



### 3.3.4 **Loiter Time—used in third spiral**

Loiter time is the additional attribute added in the third spiral. It is a large change in the bomb, changing the configuration significantly. A small turbojet, similar to the turbojet used in the LOCAAS system, allows the bomb to remain over the battlefield. A camera allows either a human operator or software recognition systems to identify a target in the battlefield. The boundaries of this system were set by comparison to the LOCAAS system, and were set at 45 minutes for the upper range and 10 minutes for the lower range. The interview scenario was:

Scenario: A new fuel is being considered by the engineers working on the third spiral of the SDB. The old fuel has a 50-50 chance of having a loiter time of 45 minutes or of 10 minutes. The new fuel has a # chance of having a loiter time of 45 minutes and a 1-# chance of having a loiter time of 10 minutes.

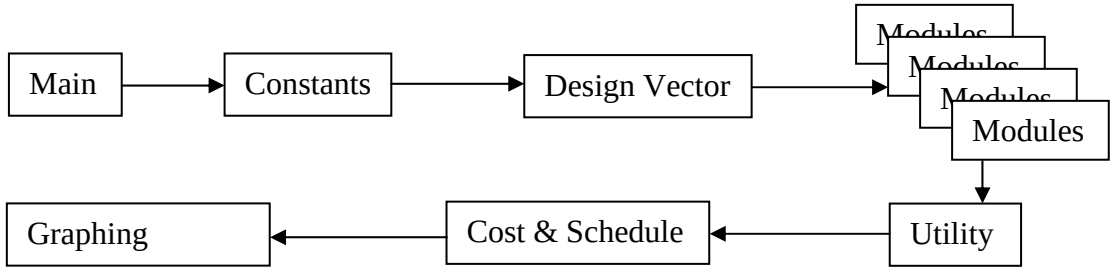
The Preference curve is presented here:



# 4 Mathematical Modeling

## 4.1 Overall Model Structure

Matlab is used to model each attribute. The full code can be found in Appendix A. It was found that most modules must be changed somewhat for each spiral; the amount of change, however, is very small. The modules are numerous; the overarching structure of the code is as follows:




## 4.2 Data storage

Data begins in the Constants module. There, all of the constants are stored into a Matlab structure file. A structure is similar to a vector, but structures can hold vectors, matrices, or text strings in each cell instead of a single or double precision numerical value.

The Design Vector module then creates the Design Vector structure. This structure is fairly large, with five or six different vectors attached to it. Each vector within a structure receives the same indices. This facilitates data retrieval, since each possible combination of the variables has a distinct index number within the Design Vector structure.

The program places all of the output variables for each architecture into another structure, titled “Tradespace.” This structure is quite large, with well over a dozen different vectors

indexed together. The Tradespace structure contains the complete description of each architecture, including individual attribute utility, multi attribute utility, cost, engineering hours, values for each attribute and several intermediate attributes.

## 4.3 Main Module

### 4.3.1 Introduction

The Matlab model begins with a central script file, Main. This script file calls all of the other models in the correct order and allows overall configuration of the code. Main can be modified to run only portions of the code as long as particular care is taken to ensure that all the dependencies are taken care of for each module.

The modules for the model of the first spiral are called in this order:

Constants
DesignVector
GlideDistance
CrossSectionalDensity
CircularErrorProbable
NumberInFA22
CostSchedule
Sortie
nirav_calculate_K
Utility

Each of these modules ascertains the size of the main data arrays, DesignVector and Tradespace, and then initiates a loop to do its calculations for each architecture under consideration.

## 4.4 Description of the model for the first spiral

### 4.4.1 Constants Module

#### 4.4.1.1 Introduction

This module sets all of the constants that are needed for the code.

#### 4.4.1.2 Required Inputs

This module requires no inputs.

#### 4.4.1.3 Output Descriptions

The Constants structure is the output of this file. It can be seen in Appendix A, Matlab code.

#### 4.4.1.4 Key Assumptions/Simplifications

There are several assumptions contained within the constants file for spiral one:

- Bombs have an average density of 0.174 pounds per cubic inch.
- Launch speed is the same for every bomb.
- Launch altitude is the same for every bomb.
- The $CD_o$ is 0.0045
- 5000 bombs will be built
- 10 bombs will be tested
- Two layers of bombs can be put into the F/A-22
- The F/A-22's bomb bays are 30 inches wide and 216 inches long

#### 4.4.1.5 Rationale for Simplifications

- Average density was determined by averaging the density of approximately 15 different bombs. The SDB is likely to have a higher density than existing bombs, since engineers will be attempting to make the SDB a small system.
- Launch Speed and altitude must stay the same so that different architectures are comparable.
- The estimated $CD_o$ is from Raymer, 3rd edition, 1999.
- The number of bombs built and tested are estimates used in the cost model. They can be changed easily; unfortunately, the author had limited access to sensitive data and could not determine these numbers.
- The bomb bay simplification into layers stacked vertically and a simple length and width were necessary due to the same limited access to sensitive data. While the bomb bay size is not classified, finding an explanation of the rack mounting system and size of the bay turned out to be nearly impossible.

#### 4.4.1.6 Fidelity Assessment

This module does not require a fidelity assessment.

#### 4.4.1.7 Verification

This module does not need verification.

### 4.4.2 **Design Vector**

#### 4.4.2.1 Introduction

The design vector enumerates all of the possible design combinations that exist.

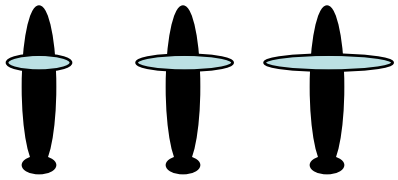
#### 4.4.2.2 Required Inputs

This module does not require any inputs.

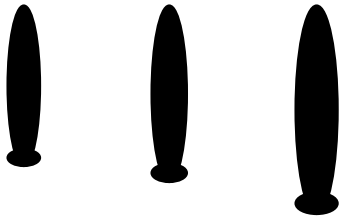
#### 4.4.2.3 Output Descriptions

In the first spiral, the design vector enumerates the following design choice possibilities. Visual aides are added to a few of these choices to assist the reader in understanding the physical implications of these choices.

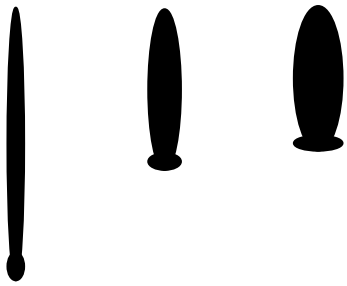
- Aspect Ratio from 0.1 to 2 in increments of 0.05 (39 possibilities)



- Explosive weight from 150 to 220 pounds in increments of 10 (8 possibilities)



- Diameter from 3 inches to 9 inches in 1 inch increments (7 possibilities)



- Guidance is set to either ‘inertial’ or ‘GPS’ (2 possibilities)

- Actuators from 2 to 3 (2 possibilities)

The enumeration of this space gives a total of 39*2*2*8*7=8,736 architectures. The architectures are contained within a Matlab structure variable, each combination having its own index number. The structure allows each design choice to be kept in a vector attached to it. Each architecture contains a single combination of these five design possibilities. For instance, the structure entry "DesignVector(1)" lists values as follows:

- DesignVector(1).AspectRatio=0.1
- DesignVector(1).Guidance='inertial'
- DesignVector(1).Actuators=2
- DesignVector(1).ExplosiveWeight=150
- DesignVector(1).Diameter=3

#### 4.4.2.4 Key Assumptions/Simplifications

This module assumes that the total weight of the bomb will consist only of the explosive weight and the casing weight. The casing weight is estimated by multiplying the explosive weight by 1/10, multiplying the result by 10 times the aspect ratio, and multiplying the result by diameter in feet.

#### 4.4.2.5 Rational for Simplifications

It is assumed that the casing weight will be affected by a mathematical relationship between the explosive weight, the wing size, and the diameter of the bomb. The particular equation was developed such that the total bomb weight fell in approximately the range that other bombs of similar size.

#### 4.4.2.6 Fidelity Assessment

Casing weight for a 6 inch diameter 175 pound explosive weight bomb with a wing aspect ratio of 1:1 is 87.5 pounds. This is approximately 30% of total bomb weight, which is well in line with average values of casing weight as a percentage of total bomb weight.

4.4.2.7 Verification

This code outputs the values appropriately. When the ranges are changed for one variable, those changes are interleaved throughout the resulting DesignVector structure correctly.

### 4.4.3 Standoff Distance

#### 4.4.3.1 Introduction

The code begins by estimating the maximum lift over drag ratio, which is equivalent to glide ratio, which allows us to approximate maximum standoff distance. The estimation equation for lift over drag is as follows:

$$\frac{L}{D}_{\max} = 0.5\sqrt{\pi * AR * e} \Big/ CD_0$$

Where AR is the aspect ratio of the wings, *e* is the Oswald efficiency factor, and $CD_o$ is the coefficient of drag at zero lift. The Oswald efficiency factor, *e*, can be determined by the following relationship:

$$e = 1.78(1 - 0.045 * AR^{0.68}) - 0.64$$

Both of these equations are taken from Raymer, 3rd ed, 1999.

The only unknown factors in the previous two equations are the aspect ratio and $CD_o$. The Design Vector is constructed to vary the aspect ratio for the wings, and the launch speed is

set in the constants module. The $CD_o$ is an estimate based on estimates in Raymer, 1st ed., is set in the constants and should be varied in sensitivity analysis.

From these equations and these variations, it is possible to assign an estimated glide distance to each architecture by simply multiplying the initial launch height by the glide ratio. After completing this calculation, the program converts the distance into nautical miles.

#### 4.4.3.2 Required Inputs

The only inputs required for this module are the DesignVector structure and the Constants structure.

#### 4.4.3.3 Output Descriptions

This module begins the Tradespace structure by adding the vector GlideDistance to the structure.

#### 4.4.3.4 Key Assumptions/Simplifications

There are several simplifications to this module. They are:

- Wind is ignored
- The ground is flat
- The Oswald Efficiency Factor is estimated
- Aspect ratio is assumed to be a sufficient aeronautical description of the bomb
- Supersonic flight, a key attribute of the SDB, is ignored

#### 4.4.3.5 Rational for Simplifications

- Wind is a random factor
- Ground shape is a random factor
- The estimating equation for the Oswald Efficiency Factor is reasonably accurate
- While aspect ratio is an insufficient aeronautical description for the bomb, it is one of the key designators of how much lift and drag the wing has
- Supersonic flight is an unnecessary complication. While it is what the bomb is designed to do, supersonic flight distance will be proportional to the subsonic flight distance. Since the utility equations simply rank architectures as better or worse than each other, but do not say by how much, this simplification causes no degradation of final data.

#### 4.4.3.6 Fidelity Assessment

This module returns glide distances within the ranges expected for the SDB.

#### 4.4.3.7 Verification

The module was verified by passing it large and small values. The module returned reasonable values even with unreasonable data, with glide distances ranging from very small (with a low launch speed, height, and aspect ratio) to very large. (With a high launch speed, height, and aspect ratio.)

### 4.4.4 Number of Sorties to Destroy Target Set

#### 4.4.4.1 Introduction

This module begins with a target set, defined in the constants file, and determines the number of bombs needed to guarantee a 95% chance of a direct hit on each target. The sum of the bombs needed is then divided by the estimated carrying capacity of the F/A-22 to determine how many plane flights will be needed to destroy the target set. For the purposes of this model, each plane flight is considered a sortie, even if it flies with other planes. Thus, if four planes flew together to destroy a target set, this would be considered four sorties.

#### 4.4.4.2 Required Inputs

First, a target set must be defined. It is unclear exactly how the SDB will be used in combat. Dr. Rebentisch believed that it would be used in Kosovo-like situations, where targets of opportunity would be of greatest importance. The surrogate user, on the other hand, suggested that it would be used to target bunkers and depots. Thus, the target set consists of both classes of targets: 5 bunkers, 2 tanks, 5 armored personnel vehicles, and 10 tanks.

Since the target set is defined in the constants, clearly the Constants structure needs to be input. The Constants structure also sets the assumed kill percentage at 95%, and the size and number of the F/A-22's bomb bays.

#### 4.4.4.3 Output Descriptions

A target is considered destroyed if there is a 95% or better chance of a direct hit. To simulate hardness and/or armor, the target size is set as follows:

Bunker—0.5 meter target radius
Tank—1 meter target radius
APV—2 meter target radius
Jeep—3 meter target radius

These radii are then multiplied by the fractional bomb size. Fractional bomb size is calculated as follows: first, the current architecture's explosive weight is divided by the largest explosive weight enumerated. Fractional bomb size is the square root of this fraction. It is necessary to take the square root of this fraction because the force of an explosion falls off as the square root of distance.

Circular Error Probable (CEP) is defined as the distance from the target within which half of the bombs will strike. Thus, if 100 bombs are dropped with a target of a tank, if the bombs had a CEP of 10 meters, 50 of them would strike within 10 meters of the center of the tank. The bombs are assumed to fall in a two-dimensional normal distribution around the center of the target.

Within this model, CEP is assigned to the SDB architectures in the following way: there are two types of guidance available, GPS/inertial and inertial only. In addition, a bomb can be configured with two or three actuators. This gives the following four possible architectures, which are assigned the following CEP:

GPS/inertial, 3 actuators—5 meters

GPS/inertial, 2 actuators—7 meters
Inertial only, 3 actuators—9 meters
Inertial only, 2 actuators—11 meters

This CEP is then used in the following way to determine the total number of bombs that is required to destroy the target set. First, we determine the probability P of a direct hit based on target radius and the normal distribution based on CEP. Bombs are always dropped in pairs.

The following equations then determine the number of bombs necessary to destroy a target:

P=Probability of direct hit
1-P=probability of miss
$(1-P)^2$=probability of a pair of bombs missing
$((1-P)^2)^{n/2}$=probability of n bombs missing when dropped in pairs
$1-((1-P)^2)^{n/2}$=probability of at least one hit when n bombs are dropped in pairs

This equation is set equal to 0.95, the percentage required for a guaranteed kill, and solved for n, then multiplied by the number of targets of that target type. After running through all different target types and summing n, the total number of bombs needed to destroy the target set is known.

Now the carrying capacity of the F/A-22 must be determined. The shape of the bomb bays in the F/A-22 are quite complex. With more time and more information, a rack mounting system could be designed and optimized, but both time and information were unavailable to this author. However, this does not detract from the usefulness of the MATE process; an engineer working with the government would have better access to sensitive information and more resources to optimize the model. In this case the number is determined in the following simple way:

First, the diameter of the bomb is used, along with an average bomb density and the configured total weight of the bomb, to determine overall length of the bomb. An assumption is

made that the rack mounting system within F/A-22 bays can carry two layers of bombs vertically. 10% of the length and width of the bay is reserved for the rack mounting system. Knowing these parameters, the number that fit into a simplified rectangular bay is easily determined.

The total number of bombs needed to destroy the target set is then divided by the carrying capacity of the F/A-22 to determine the total number of sorties needed to drop that number of bombs.

#### 4.4.4.4 Key Assumptions/Simplifications

There are many assumptions in this particular attribute calculation. They are:

- The target set is set at a specific number of buildings and vehicles
- The target size approximation meant to account for target hardness
- Target is assumed destroyed when there is a 95% chance of a direct hit
- Bomb effectiveness affects the target size, i.e., the difficulty of destruction is increased for a smaller bomb by reducing the target size
- The F/A-22's bomb bay dimensions are set as a rectangular solid

#### 4.4.4.5 Rational for Simplifications

- The target set is simply to rank the different effectiveness of the architectures under consideration; so long as the target set is the same across all bombs, the ranking should be accurate
- A target's hardness makes it more difficult to destroy. Making the size of a direct hit smaller makes the target more difficult to destroy. Both difficulties are overcome by dropping more bombs; an engineer with more information could set each target's size to correlate with known figures. The code may be easily changed to reflect such information. In addition, since the targets are the same across each bomb architecture, and the point at hand is to rank the bomb architectures, it should be an accurate simplification.
- Probability is calculated in such a way that there can never be a 100% chance of a direct hit. 95% is an approximation, and is consistent for each bomb architecture.
- To vary the architecture and see how many bombs will fit into the bomb bay, explosive size is varied. This must also affect the bomb's effectiveness. To easily model this difference in effectiveness, all the explosive sizes are normalized to the largest explosive size used and the fractional explosive effectiveness reduces the direct hit radius and thus makes the target more difficult to destroy with smaller bombs. A more reasonable

physical model could be constructed, but this simplification is a good first order approximation of bomb effectiveness based on explosive weight.

- Again, the bomb bay information, while not classified, is difficult to get and very complex in nature. Rather than optimize bomb placement, it was assumed that if the same shape and size were used for every bomb architecture, a proper ranking order would be achieved.

#### 4.4.4.6 Fidelity Assessment

This module is fairly coarse in fidelity. However, it does an adequate job for the task at hand: ranking thousands of architectures based on relative ability to achieve the user need of sending fewer planes in to destroy a target set.

#### 4.4.4.7 Verification

A special software module was written to feed this module different values to determine if any errors had been made. One error was found in the calculation of bomb length where a conversion had been neglected. Once this error was corrected, the model performed admirably, outputting reasonable data based on the inputs given it.

### 4.4.5 Cost and Schedule Model

#### 4.4.5.1 Introduction

The purpose of a cost model in the MATE process is to provide a metric to compare the estimated expense for each of the architectures. To accomplish this, a parametric cost model is used. This allows each architecture to be slightly different and still have a different cost.

In this case, a standardized cost model is unavailable. Unfortunately, cost models for weapons systems such as this one are classified or otherwise unavailable. In past MATE applications, cost models were relatively available. In extending MATE beyond space systems, as in this case, it becomes paramount to find or develop a cost model.

The cost model used here has been appropriated from the DAPCA-IV airplane cost model. It is not the most accurate cost model, nor is it the most appropriate for calculating cost on a weapons system. However, the DAPCA-IV model is the most widely applicable model available, and most of its inputs were already calculated by the model. The missing inputs were: the bombs tested and the number of bombs produced in the total production run. These are arbitrarily set in the constants structure at 10 and 5000 respectively.

This cost model did not include a good way of including the various costs for the guidance systems, so a guidance system cost model was built. Using the various tailfin kit costs and parameters, it was determined that each tailfin actuator would cost $4,000 and adding GPS to a system would cost another $3,000. This guidance cost is added to each bomb based on its configuration.

#### 4.4.5.2 Required Inputs

This model requires the tradespace, design vector, and constants structures as inputs.

#### 4.4.5.3 Output Descriptions

The model adds TotalCost and TotalHours, both vectors, to the Tradespace structure.

#### 4.4.5.4 Key Assumptions/Simplifications

As in all cost models, the assumption is that cost and schedule are driven by a few key attributes. By accumulating historical data on these attributes, one can predict future cost. This is not necessarily the case; it is hoped that the relative cost predicted by the model is indicative of actual relative cost, however, so this paper will discuss only normalized cost.

#### 4.4.5.5 Rational for Simplifications

One must estimate cost to compare the costs of different architectures in MATE analysis. It is also necessary to estimate costs to gain congressional approval for projects. Budgeting would be impossible without such mathematical estimations of cost as well.

#### 4.4.5.6 Fidelity Assessment

As with any cost model, fidelity is not terribly great. For this reason, all analysis is made on normalized results from the cost model. In this case, it is the relative cost or schedule that interests the user more than the absolute cost or schedule.

#### 4.4.5.7 Verification

A special software module was written to feed this module different values to determine if any errors had been made. Several small errors were discovered and repaired, after which the output was commensurate with the input that created it.

## 4.5 Description of the model for the second spiral

### 4.5.1 Constants

This module remains unchanged for the second spiral, an example of 100% code reuse between spirals

### 4.5.2 Design Vector

The enumeration of the design vector is identical for the second spiral. This is another example of 100% code reuse between spirals.

### 4.5.3 **Retarget Time**

While the addition of retarget time does not change the model of the physical bomb itself, it adds a new capability: retarget time, the minimum time in seconds before the bomb strikes its original target that the bomb can be given a new target.

Retarget time does not change either of the other two attributes; it simply adds a new attribute that is valued independently of the other two. It is up to the user to decide which of these attributes are most valuable to him or her.

Of course, in reality, the physical bomb would need a small change or two to be able to retarget: it would need a receiver put into it to receive the retargeting command, and possibly a new nosecone containing a camera and transmitter. These are small parts of the system and, along with the programming challenges that such retargeting would pose, are ignored in this model. The cost of such changes has been reflected in the cost model by adding a delta to the cost of each architecture.

#### 4.5.3.1 Introduction

Retarget time is the smallest time before impact, in seconds, in which the bomb can be accurately retargeted to a new target up to 20 degrees away from the bomb's current vector of forward movement. This is a trade that shows how flexible the MATE process is, for this trade is an addition to the bomb that barely changes the architectural models, and yet shows differing utilities based on users asking an existing design to do more.

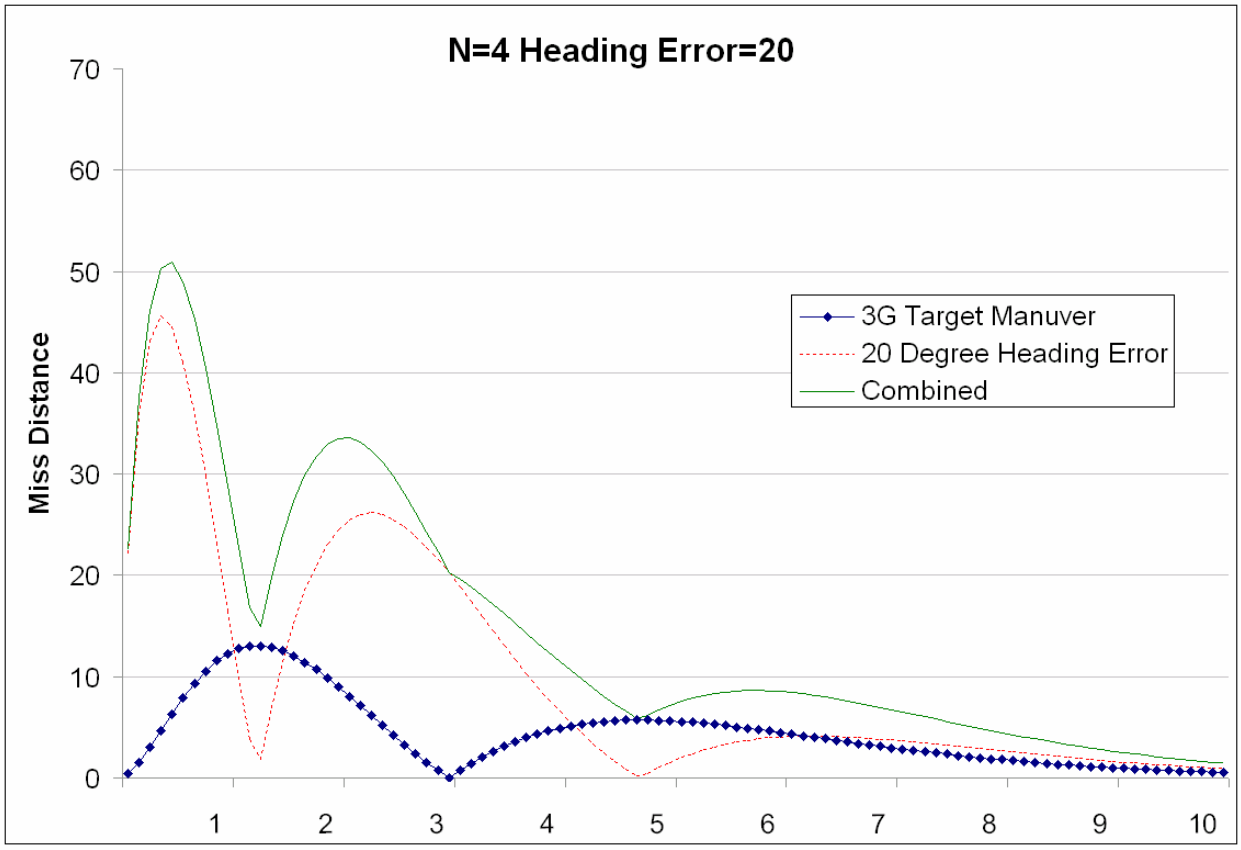
#### 4.5.3.2 Required Inputs

This module requires the Constants structure, the DesignVector structure, and the Tradespace structure.

#### 4.5.3.3 Output Descriptions

To get proper output for this model, a physical model of the system was made. The method of adjoints was used by modifying fortran code developed in *Tactical and Strategic Missile Guidance*. This code allowed the creation of graphs and spreadsheets of data showing miss distance based on when a 20-degree Targeting Error was discovered.

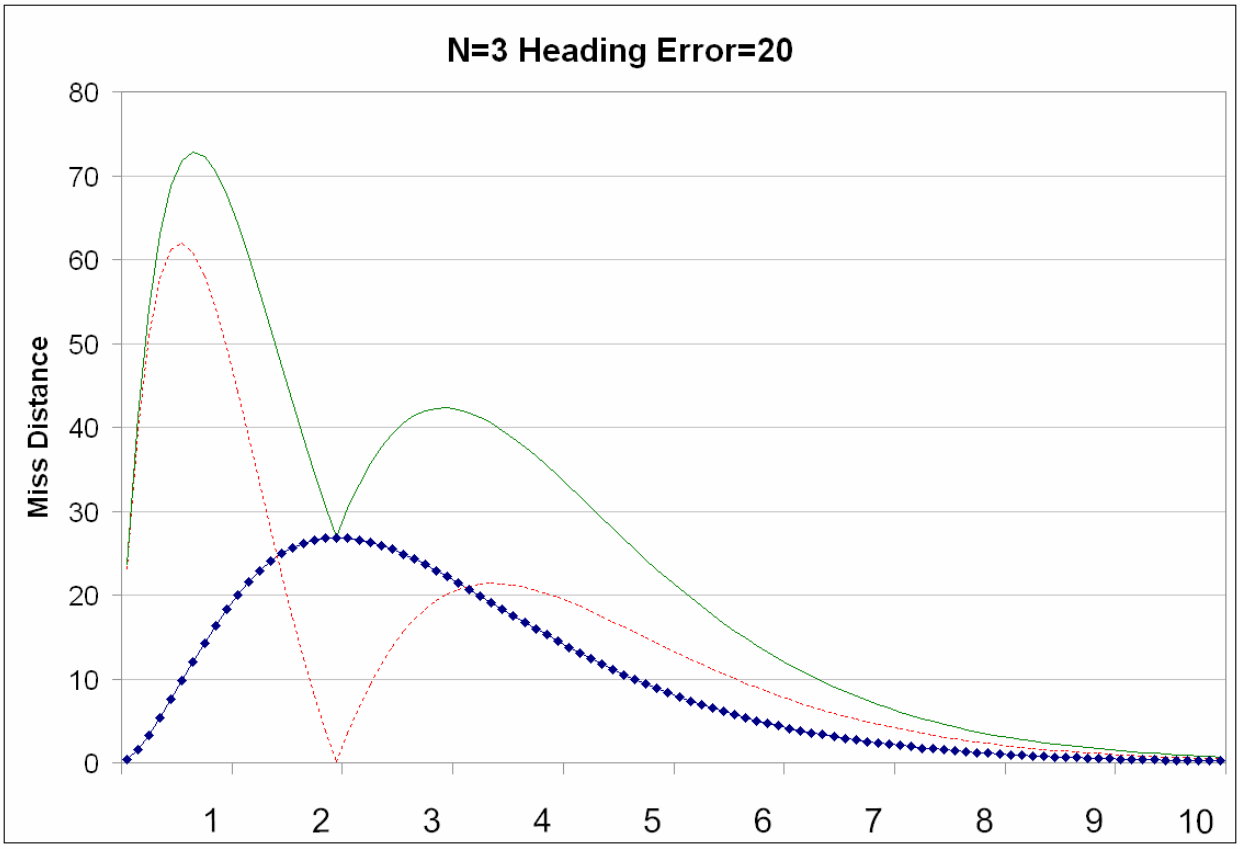
The resulting four sets of data show miss distance as a function of seconds before Retarget. The following graphs were then generated:



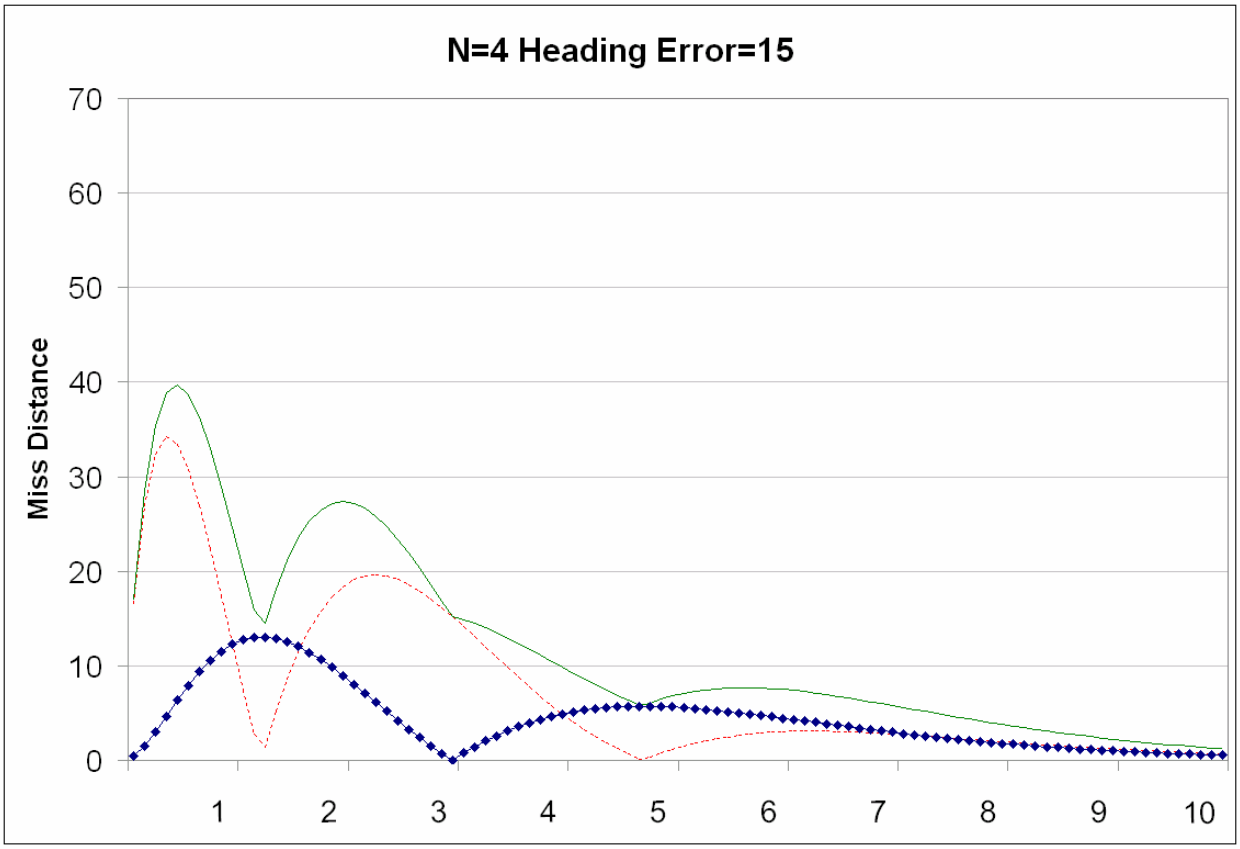

This graph shows miss distance in feet based on the number of seconds before impact the bomb is retargeted for a bomb architecture containing three actuators and inertial guidance only.

For instance, if this bomb configuration is retargeted three seconds before impact at a stationary target, it bomb will miss the new target by approximately 20 feet. If it is retargeted a half a second before impact, it will miss its new target by approximately 45 feet. If the bomb is

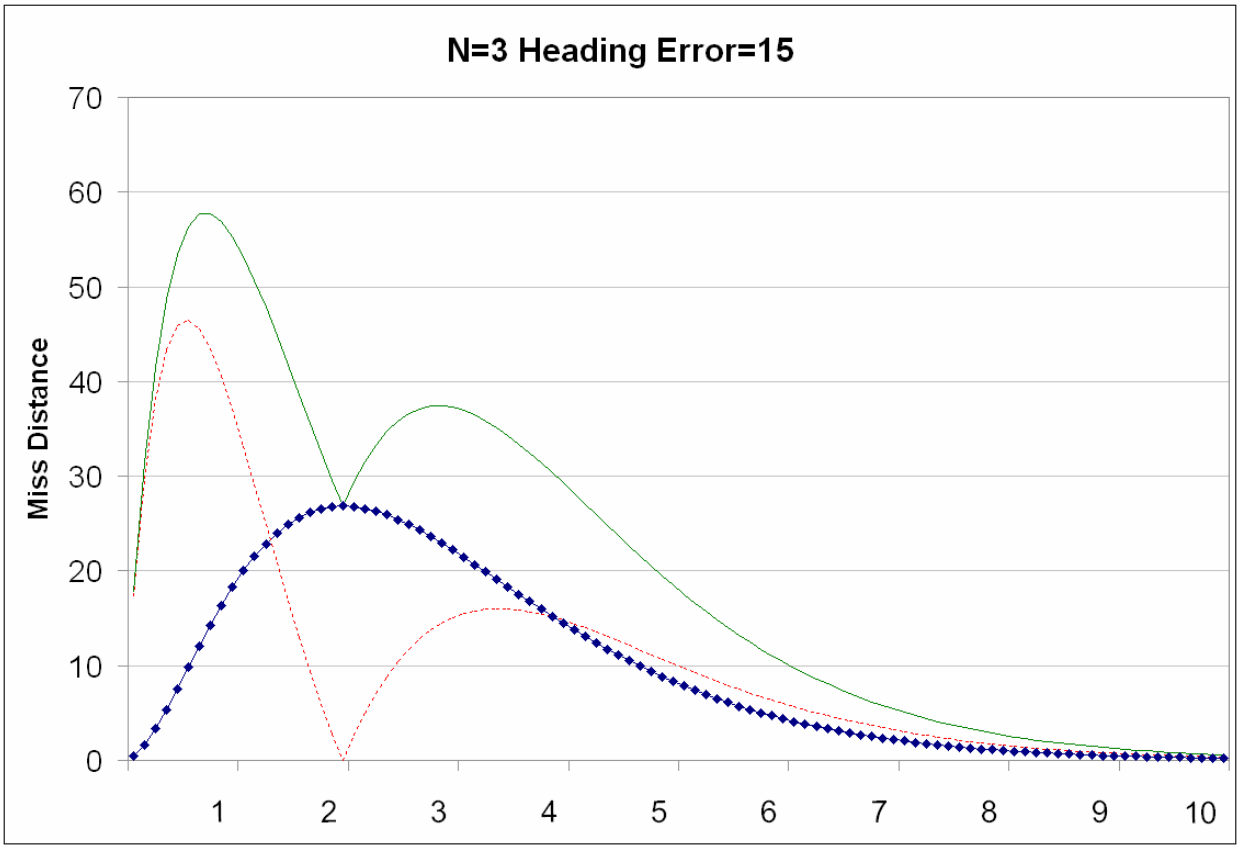
tracking a moving target that makes a 3g maneuver five seconds before impact, then the bomb will miss its target by approximately 6 feet.




This graph shows miss distance in feet based on the number of seconds before impact the bomb is retargeted for a bomb architecture containing two actuators and inertial guidance only. Note that the curves are significantly higher for fewer actuators.




This graph shows miss distance in feet based on the number of seconds before impact the bomb is retargeted for a bomb architecture containing three actuators and GPS guidance. Note that the curves are significantly lower for GPS and higher actuators; however, since the expected CEP is so much smaller for this architecture, it actually has a higher (worse) retarget time than the inertial, 3 actuator system.




This graph shows miss distance in feet based on the number of seconds before impact the bomb is retargeted for a bomb architecture containing two actuators and GPS guidance. Note that the curves are significantly higher for the lower number of actuators despite having a GPS system installed.

#### 4.5.3.3 Key Assumptions/Simplifications

Some modeling assumptions were made in applying this information to the model:

First, Targeting Error was used as a surrogate for a retarget command. Thus, instead of the model determining how much of a miss would occur based on a launch X seconds before impact with a Y degree targeting error, the model is used to determine how much of a miss would occur based on a retarget command with the same X seconds before impact with the same Y degree retarget standing instead of the targeting error.

Second, the *guidance ratio* had to be set. This is a quantity in the physical model represents how much force is actually applied by the bomb control surfaces when a certain amount of force is requested by the guidance system. For a more thorough description, see Zarchan's Tactical and Strategic Missile Guidance, AIAA 2002. As a higher number represents a better system, and, due to the mathematics involved in this particular model, 3 is the minimum guidance ratio, the guidance ratio was set to 3 for the two actuator SDB configuration and 4 for the three actuator system. These fall within the range of 3-5 for typical guidance ratio numbers given by Zarchan 2002.

Another assumption was that the inherent accuracy of the inertial only guidance system was significantly less than the accuracy of the GPS and inertial system. This was represented by reducing the Targeting Error by five degrees for the GPS system. Thus, the Retargeting Error was set at 20 degrees for the inertial guidance system and 15 degrees for the GPS and inertial guidance.

Another assumption is that the miss distance is used as a surrogate Circular Error Probable distance. The physical model does not take into account many of the things that affect bomb flight, such as wind speed and direction. To generally account for these effects, however, it is assumed that a miss of approximately 9.5 feet is actually the mark within which half of the bombs will strike, given that they are used in the identical situation.

Finally, an assumption was made about the system itself. It is assumed that both retargeting and target tracking are possible for the SDB system. No exceptional effort was made to determine the amount of programming that would be required or what sort of modular architecture would be needed, such as a transceiver, antenna, data link, or other additional electronics.

#### 4.5.3.4 Rational for Simplifications

Targeting error used as a surrogate for a retarget command is a valid assumption. The forces on the bombs are similar, with the only exception being that a bomb is moving at a slightly different speed at launch. This difference is very small.

Guidance ratio is a way of differentiating the different number of actuators in the system. It is reasonable that, everything else being the same, more actuators can apply more force in the system.

The assumption of assigning the miss distance into a Circular Error Probable normal curve is meant to capture many factors not included in the physical model.

The assumption of GPS being more accurate than inertial only systems is an obvious one. The choice of the specific value of 25% more accurate (15 error as opposed to a 20 degree error) was made with the knowledge that architectural ranking is what this model is supposed to do. With more knowledge of guidance and control, a better model could be made and used, but due to time constraints, this approximation was made.

#### 4.5.3.5 Fidelity Assessment

This code does not approximate wind interference, possible jamming efforts, incorrect target coordinates, or any one of a number of other effects that could make the bomb less accurate. However, the assumption of assigning the miss distance into a Circular Error Probable normal curve makes these simplifications less important. The code should be quite accurate, given this assumption, but the author has no way to test or even estimate the possible error of this module.

#### 4.5.3.6 Verification

Another module was written to verify this code. This sub module passed varied inputs and the outputs were checked for validity. No out of the ordinary outputs were observed.

## 4.6 Description of the model for the third spiral

### 4.6.1 Constants

This module sets all of the constants that are needed for the code. The following constants are added in the third spiral:

```
Constants.SpecificFuelConsumption=0.8;   %pounds per hour per pound,
                                         estimated from table 3.3 in Raymer
Constants.MaxThrust=30;                  %pounds of thrust; estimated from
                                         low LOCAAS number
Constants.MaxMach=0.99;                  %entire model is subsonic
Constants.TurbineInletTemperature=70;    %Degrees F
```

### 4.6.2 Design Vector

A new vector, titled “FuelWeight” is added to the design vector structure file and is varied from 10 to 25 pounds in increments of 5 pounds. This increases the tradespace to a total of 34,944 architectures. It is assumed that the bomb is “stretched” a bit to accommodate the fuel.

### 4.6.3 Loiter Time

#### 4.6.3.1 Introduction:

Loiter time is the time in minutes that the bomb can stay aloft over the area of interest. This ability is similar to the LOw Cost Autonomous Attack System (LOCAAS) system and may or may not be utilized as the third spiral of the SDB system.

The SDB third spiral is presumed to use a small, 30 lb. thrust turbojet engine to power its flight. This engine, and a fuel tank, are added to the existing system and expected to increase the weight of the system somewhat. The fuel weight is varied in the DesignVector module, as mentioned above, but the weight of the turbojet is small enough (3-5lbs—SWB produces a model turbojet engine, the SWB 11 Mambo, that provides nearly 12 pounds of thrust and weighs

only 1.9lbs) to ignore. The cost impact of this engine is added in the DAPCA IV cost model, which has a subsystem cost associated with turbojet engines.

#### 4.6.3.2 Required Inputs

This module requires the Constants structure, the DesignVector structure, and the Tradespace structure.

#### 4.6.3.3 Output Descriptions

Loiter time is calculated through the use of the Breguet Range Equation:

$$\text{Time in hours} = \left( \frac{L}{D_e} \Big/ SFC \right) * \ln\left( \frac{W_i}{W_f} \right)$$

Where $L/D_e$ is the lift over drag for maximum endurance, SFC is the specific fuel consumption, $W_i$ is initial weight and $W_f$ is final weight.

First, initial weight is determined by adding fuel weight, casing weight, and explosive weight. Final weight is the addition of casing weight and explosive weight. Specific fuel consumption is estimated from a table in Raymer, and the endurance lift over drag is 0.886 * maximum lift over drag, which has already been estimated.

It is a simple matter to have the computer calculate this for each of the 34,944 architectures, multiply by 60 to convert from hours to minutes of time aloft, and add this as a vector onto the tradespace structure.

#### 4.6.3.3 Key Assumptions/Simplifications

- The weight of the turbojet is ignored
- The configuration of the bomb is assumed to be simply “stretched” to add the fuel tank
- The Breguet range equation is assumed to be sufficient to determine time aloft.

#### 4.6.3.4 Rational for Simplifications

- The weight of the turbojet ranges from 1-2% of the various bomb configurations and will not significantly affect the size of the bomb.
- This model is an architectural one, and it is appropriate to assume that the geometry of the bomb is modular enough to add a fuel tank. If the MATE procedure were done in advance of the spirals, it might be possible to use the knowledge and insight gained by modeling several spirals ahead to help design the system for flexibility from the start.
- The Breguet range equation is an appropriate approximation at this level of design.

#### 4.6.3.5 Fidelity Assessment

This module is accurate enough for the ranking of architectures. The inaccuracies come from the estimation of SFC, the max range L/D, and the Breguet range equation itself. Unfortunately, the author has no methods at hand to numerically determine the inaccuracy of these assumptions.

#### 4.6.3.6 Verification

This module was tested by passing it very high and low numbers for L/D and SFC. While some of the numbers returned seemed ridiculous at first glance, after checking these numbers with hand calculations, they were determined to be in the proper range. The loiter times in the hundreds and even thousands of minutes were explained when it was realized that the L/D passed was higher than physically possible and the SFC was only 1/10 the predicted value.

# 5 Findings

## 5.1 First Spiral

The code has been written, the interviews have been completed, but what does all of this do for us at the end? MATE's main use is in producing tradespaces in which thousands of architectures can be compared to each other in various ways, usually by utility and cost. There is also the interview data to explore, including the all-important k values, the values that weight each of the attributes.

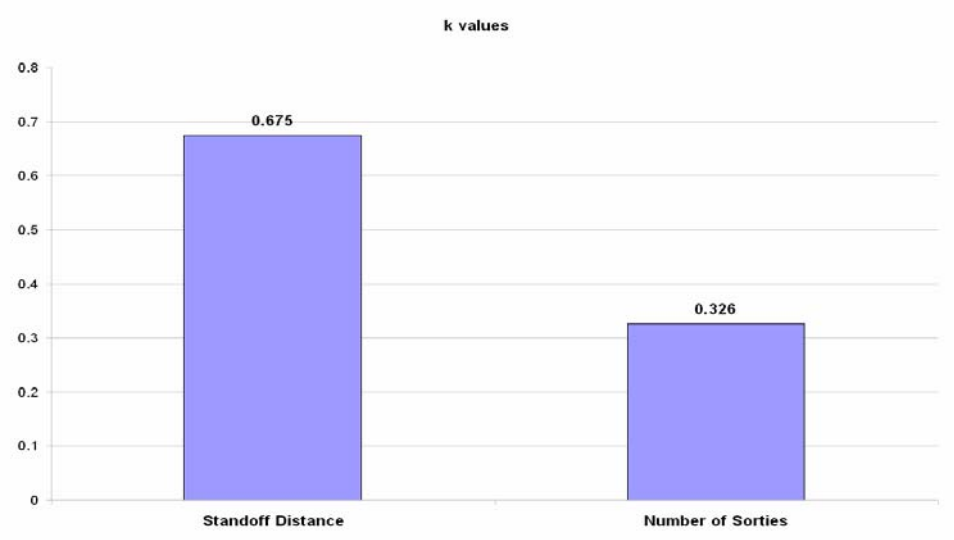
It is to be noted that the following charts, graphs, and tables generally represent only the utility and cost of the system; this is due to multi-attribute utility being a single metric that sums up the utility of the architecture to the user; in some ways, one can see the underlying physical configuration of the bomb as only of secondary importance. The user does not care, for instance, if the bomb is painted olive green or dark brown, whether or not the bomb is square or round; he or she just cares that the bomb is able to complete its mission successfully.

This represents a change in kind for engineers, who are used to speaking in technical terms. However, the users of systems rarely care about the specific technical numbers, they care about the performance of the system. In the case of MATE, the technical performance is measured by the multi- and single-attribute utility numbers, so they are what is generally presented here.

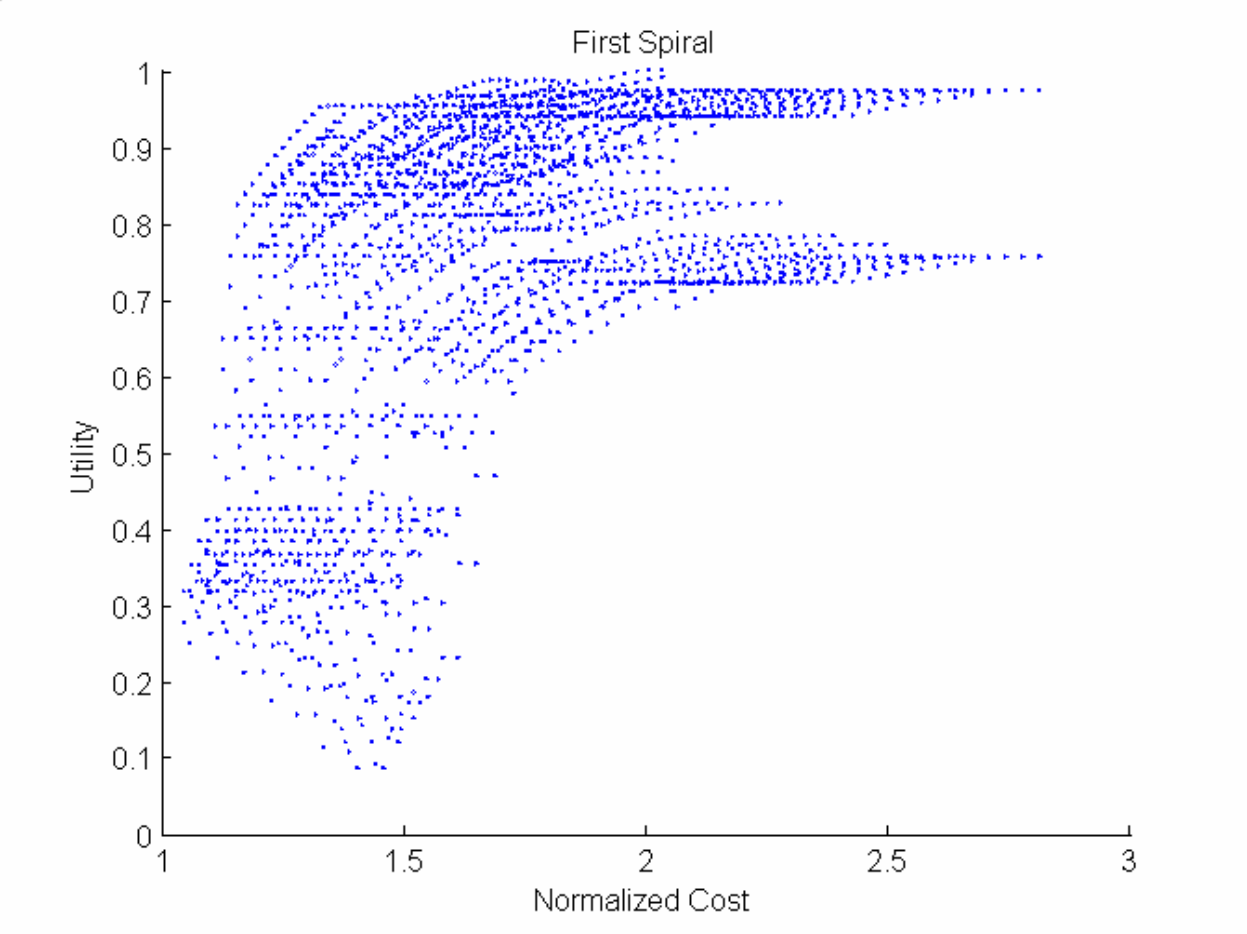
### 5.1.1 k-Values

The k values are the "attribute weighting factors." They are part of the multiattribute utility function, and help us determine which attributes are more valuable to the user. It is

important to realize that, since the function is a product function and not an additive function, these values are not directly comparable; while it is possible for us to say from the table below that standoff distance is more important to the user than the number of sorties, we are not able to say that it is twice as important. The values for the first spiral are:




It will be interesting to see how these values change when other attributes are added. The overall Tradespace from the first spiral is presented here:

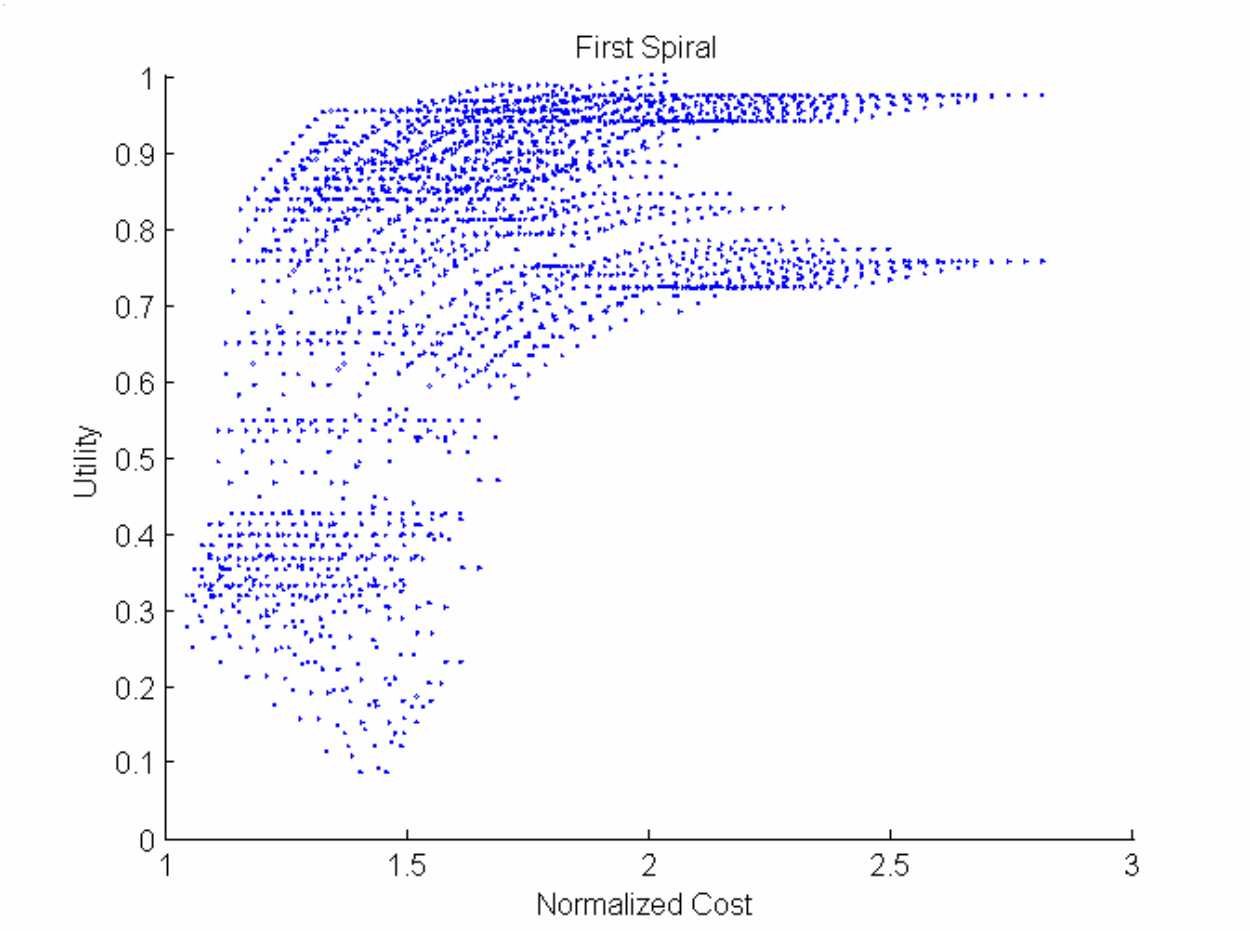



There are several ways to evaluate these tradespaces, but the three primary ways of doing so are: a) by closely examining the overall tradespace; b) by sizing or coloring the dots according to the values of one of the attributes; or c) by selecting individual architectures and looking at the details of each. We will do all of these methods, beginning with the overall tradespace and what insights we can glean from analyzing it. In this paper we will use the graphs to show only comparative values; tables that investigate specific values will follow the selected individual architectures.

### 5.1.2 **Overall Tradespace**

The tradespace of the first spiral is as follows. Each dot on the following graph represents the utility and lifetime cost of building 5,000 bombs of a specific architecture. Each dot is a

unique combination of the different possible values for the design. It is important to note that the cost in all of the plots presented in this paper are normalized to emphasize that estimated costs at the architectural level should be seen as relative costs. This is especially true in this case, as the cost model is DAPCA IV, a widely applicable airplane cost model.



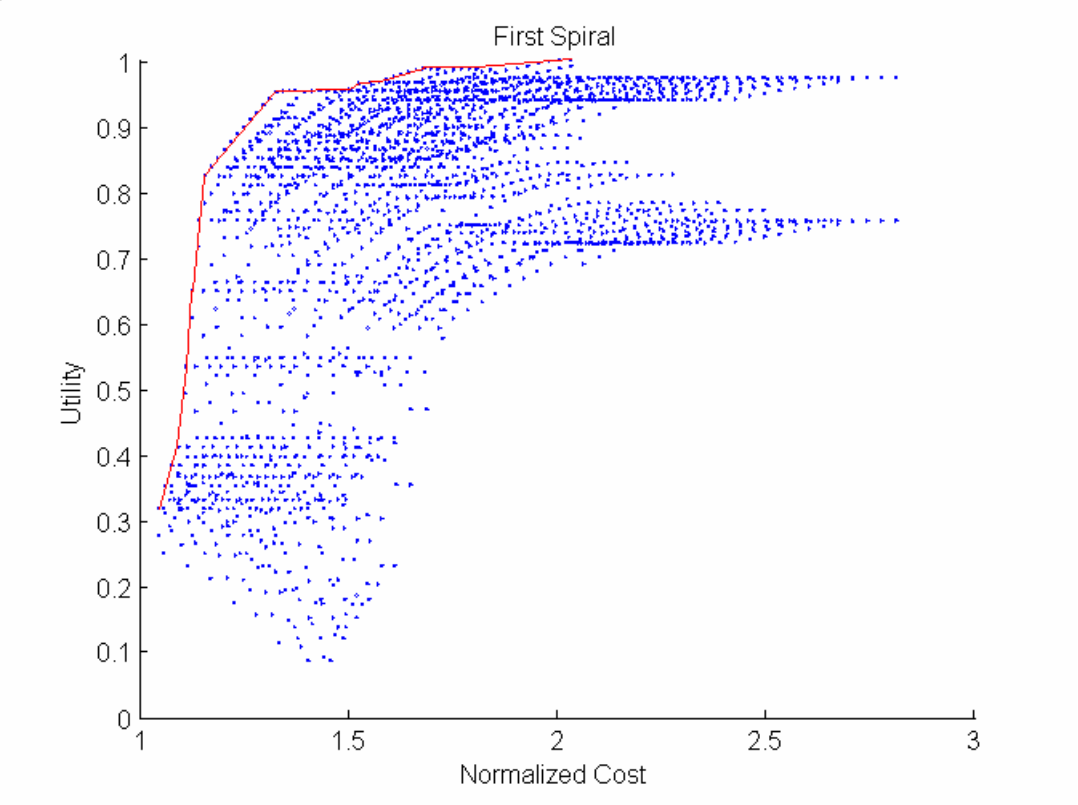

Interpreting a graph like this is never easy. Despite this difficulty, there are several things that we can do to gain insight. However, we must first reach an understanding of the utility scale, for it is far too easy to believe that the utility scale is a scale of normal numbers that can be compared normally; in fact, the utility scale is an ordered metric scale.

We are familiar with at least one ordered metric scale: temperature. We know that 15 degrees is cooler than 30 degrees, but we do not know exactly what that means; certainly, it is

inappropriate to say that 30 degrees is “twice as hot” as 15 degrees. This is similar to the utility scale; we do not know that 0.5 utility is twice the utility of 0.25, we simply know that 0.5 is better than 0.25.

Actually, we know more from temperature’s scale than we do from utility: we know that the difference between 15 degrees and 30 degrees is the same as the difference between 30 degrees and 45 degrees. We cannot even say that about utility. The difference between 0.25 and 0.5 on the utility scale is not necessarily the same as the difference between 0.5 and 0.75. This difficulty is not as great as it seems. As long as we keep in mind the nature of the ordered metric scale, we can look at the architectures as being better or worse utility.

There are a lot of architectures in this tradespace that reach almost full user utility, but there are many thousands of architectures that are “dominated solutions,” i.e., they are both more expensive and have less utility than other architectures. There is a set of architectures that forms the “pareto-optimal frontier,” i.e., another architecture can have more utility, but only at an increased cost; or another architecture can be less expensive, but only by decreasing utility. The following graph highlights the pareto-optimal frontier with a red line:



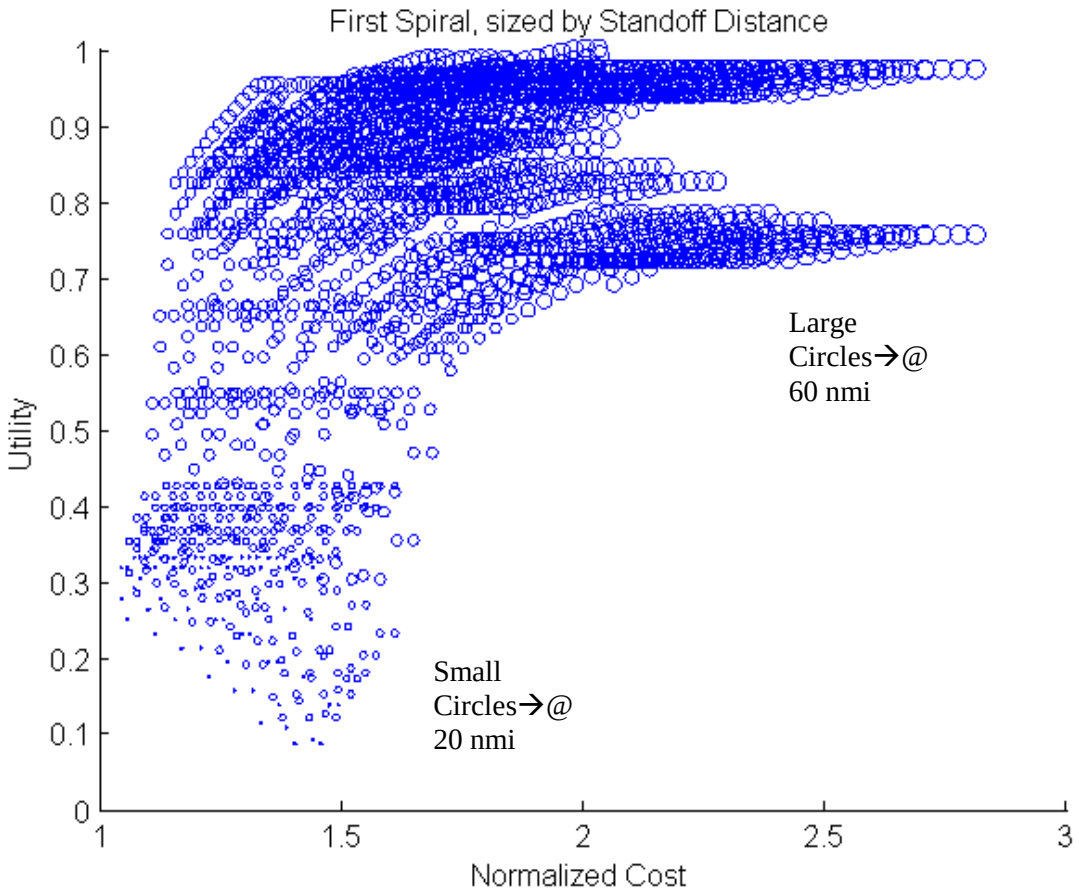

This red line is useful for determining which architectures to investigate more thoroughly. It is usual to examine those architectures which occur at the "knees in the curve," the place where the pareto-optimal line changes slope significantly.

### 5.1.3 **Analysis by Attribute and/or Design Vector**

First, time should be spent with the entire tradespace. Much insight into the system can be gained by changing the size of each dot to represent the value of one attribute for each architecture (or for interesting parts of the Design Vector) and seeing how the attributes form and affect the tradespace. When a graph shows a clear size pattern, then the attribute in question may be a primary driver of the system. When the size is more random, the attribute is less important; however, the substructure of the graph can help uncover the secondary and tertiary drivers of the system.

#### 5.1.3.1 Standoff Distance

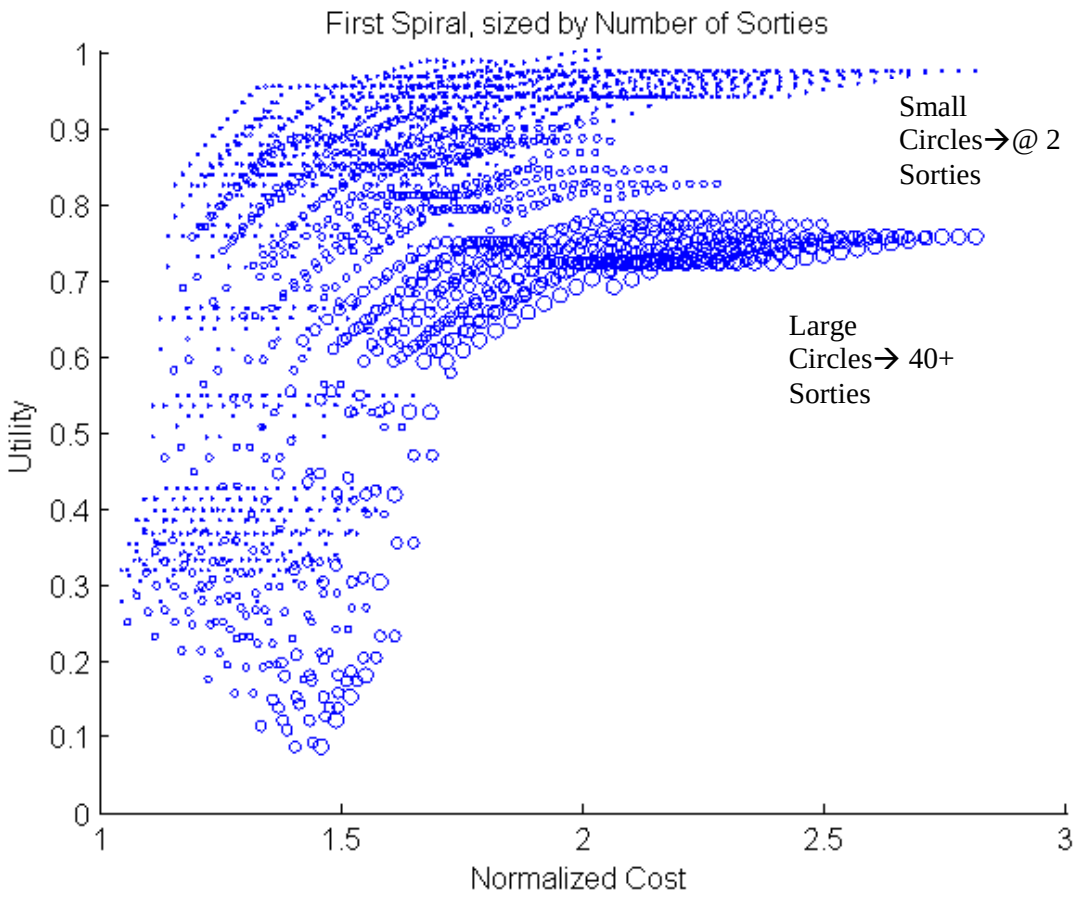
Let us look at the tradespace when sized by standoff distance. In all of the following graphs, it should be noted that the smaller the dot, the smaller the numerical value of the attribute in question. In the case of standoff distance, higher is better, and therefore larger is better.




There is a clear pattern here: small standoff distance at the bottom of the graph and large at the top. We can see here that for an architecture to reach a high utility, it must have a high standoff distance. Increasing standoff distance also seems to add cost near the top of the graph, whereas it doesn't seem to add much cost at the bottom of the graph. This might be due to the increasing complexity of the wing, represented by its weight in the models.

5.1.3.2 Sorties Required to Destroy Target Set

Here we see the tradespace colored by the number of sorties needed to destroy the target set. In this case, a LOWER number is better, and therefore the smaller architectures are more desirable:



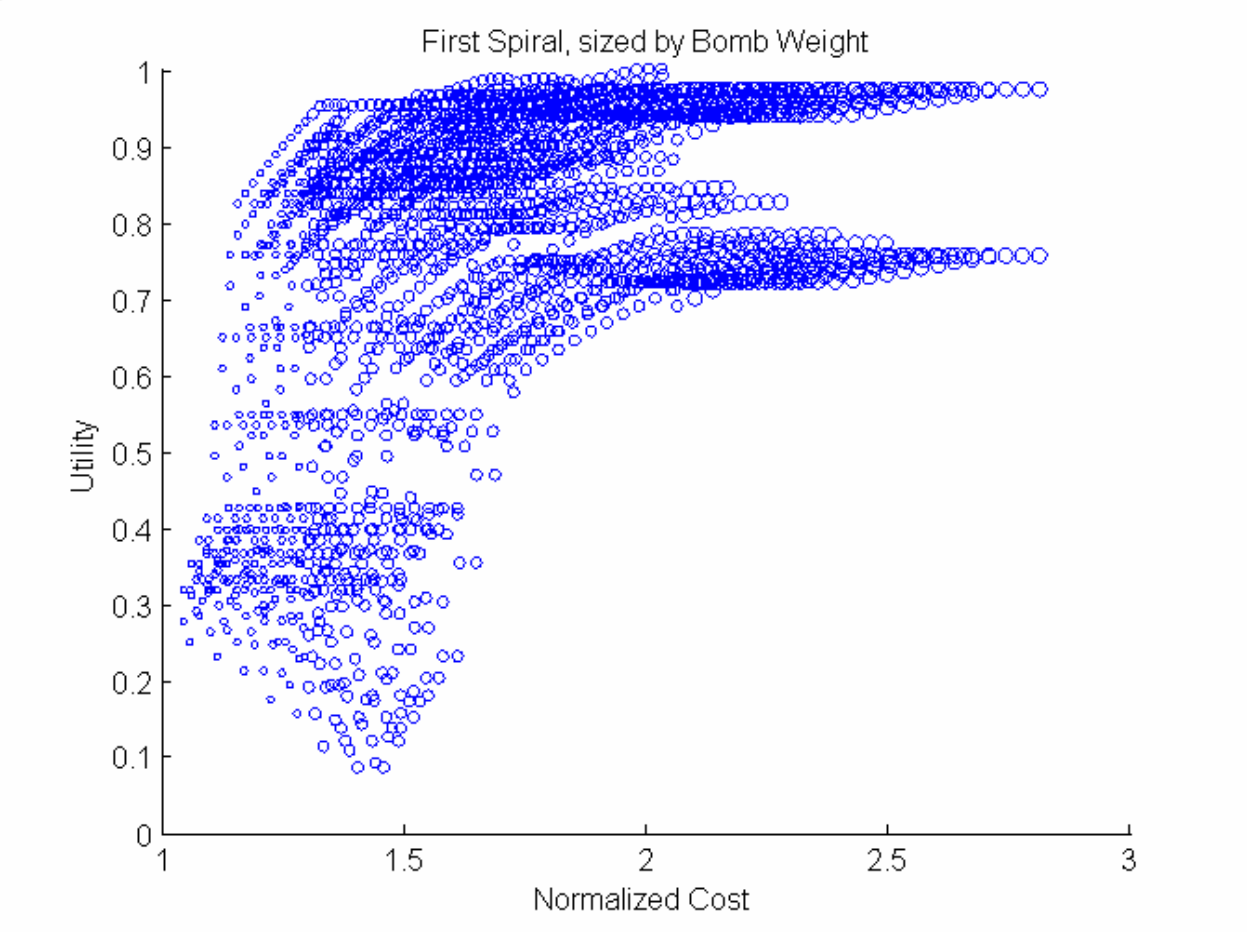

This graph shows a clear progression from lower utility and higher cost architectures having a large number of sorties to a higher utility and lower cost having a smaller number of sorties. It is interesting to note that it is impossible to be on the pareto-optimal frontier unless the number of sorties is very small; this is clearly a driver of the system at this level.

This pattern is due to several things in the models, but especially in the cost model. The number of sorties increases if the number of bombs that fit inside of the F/A-22's bomb bay is smaller. Thus, the number of sorties generally increases if the bomb is larger, and the cost model

is primarily based on two things: guidance system and bomb weight. Thus, a larger number of sorties is likely due to a larger bomb, and a larger bomb costs more.

5.1.3.3 Bomb Weight

To see whether or not bomb weight is the only driver of the number of sorties, the following plot was made. It sizes the dots of the tradespace by total bomb weight, a combination of two of the attributes in the DesignVector, explosive weight and casing weight. The total bomb weight, along with bomb diameter, determines the size of the bomb, and therefore also determines how many of these bombs will fit into the F/A-22's bomb bay.



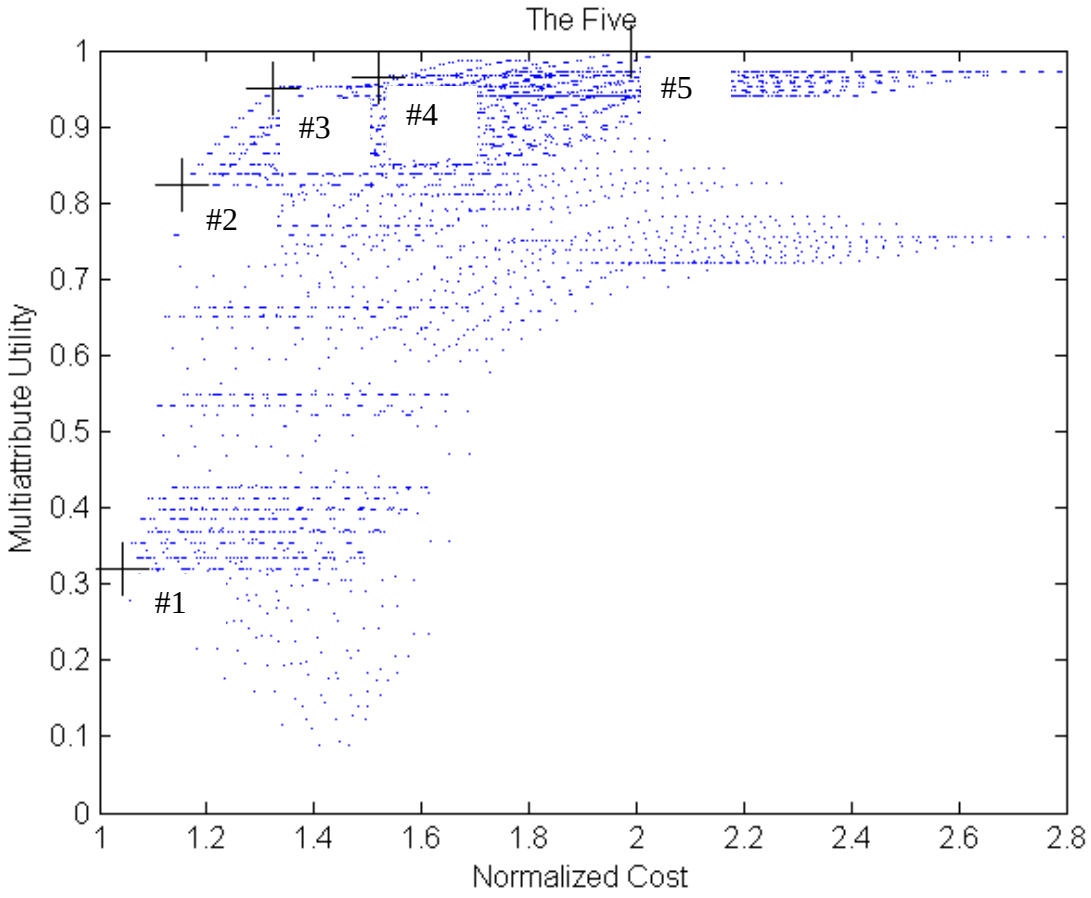

Since the bomb weight generally increases from left to right, this graph shows us that bomb weight is a primary cost driver. However, since they are not of uniform weight on the pareto-

optimal frontier, it is not simply bomb weight that drives the number of sorties. Instead, as we know from the description of the code, it is a combination of bomb size and bomb effectiveness.

### 5.1.4 **Analysis by Specific Architectures**

Now that the total tradespace has been examined, the next step is to delve more deeply into a few choice architectures. We have determined a set of architectures that have utilities which are optimal, the pareto-frontier. All that needs to be done is to choose several architectures within the pareto-frontier and break them down into their parts, and show them to the user or decision maker. He or she can then choose an architecture for detailed design.

For our purposes, we will choose five architectures, based on their location on interesting parts of the pareto-optimal front, to break out into their constituent parts, and we will track them through all three spirals to see where they end up. These five are chosen because they are either at key knees in the curve, or in large sections of equal cost but varying utility. Here are the five architectures, graphically and in constituent parts:




These five architectures are broken down in the table below:

| | Arch. # | Multiattribute Utility | Cost (Normalized) | Standoff Distance (nmi) | Standoff Single Attribute Utility | Sorties Required to Destroy Target set | Sorties Single Attribute Utility |
|---|---|---|---|---|---|---|---|
| 1 | 451 | 0.3199 | 1.0446 | 20.52 | 0.0111 | 3 | 0.9583 |
| 2 | 2019 | 0.8239 | 1.1545 | 33.62 | 0.7588 | 3 | 0.9583 |
| 3 | 4483 | 0.9512 | 1.3226 | 46.83 | 0.9476 | 3 | 0.9583 |
| 4 | 4486 | 0.9647 | 1.5184 | 46.83 | 0.9476 | 2 | 1 |
| 5 | 8071 | 1 | 1.9896 | 60.42 | 1 | 2 | 1 |

Additionally, these architectres' physical characteristics are in the following table:

| | Arch. # | Aspect Ratio | Guidance Type | Number of Actuators | Explosive Weight | Casing Weight | Diameter |
|---|---|---|---|---|---|---|---|
| 1 | 451 | 0.20 | Inertial only | 2 | 150 | 12.5 | 5 |
| 2 | 2019 | 0.55 | Inertial only | 2 | 150 | 34.4 | 5 |
| 3 | 4483 | 1.10 | Inertial only | 2 | 150 | 68.8 | 5 |
| 4 | 4486 | 1.10 | Inertial only | 2 | 150 | 110.0 | 8 |
| 5 | 8071 | 1.90 | Inertial only | 2 | 150 | 213.8 | 9 |

For illustration, if the user decided to choose the least expensive architecture as a point design to further develop, we can examine that architecture much more thoroughly: architecture #451 has wings that are only 1/5 as wide as their chord length. This means that if the wings were 10 inches from leading edge to trailing edge, they would only be two inches wide. Clearly, this would not give much lift, and this explains the small glide distance, seen in the first table's entry for architecture #451—the bomb can't stay in the air for very long if the wings don't generate much lift.

The type of guidance is inertial only, not inertial and GPS. This is one of the reasons why the cost on this architecture is so low. GPS guidance costs several thousand dollars more per bomb built, increasing the lifecycle cost significantly. In fact, this is why all of the architectures chosen are inertial only.

For the same reasons, the number of actuators is at two for our architecture #451 and for the others that we have chosen: adding another actuator (the actuator is the mechanical part that moves the fins for guiding the bomb) adds expense to the individual bomb, which drives up the cost for our run of 5,000 bombs that the cost model assumes. Because of the size of the run, adding a small cost such as an actuator, can have large repercussions on cost, whereas the accuracy impacts the number of sorties only as one of several factors.

The explosive weight is 150 pounds in architecture #451. This affects many things in the model and use of the bomb: first of all, the bomb is smaller because the explosive weight is

smaller. This helps more of the bombs fit into the F/A-22's bomb bays. Second, the amount of explosive affects the explosive power of the weapon, accounted for in the "sorties to destroy target" set module. A smaller explosive weight necessarily needs more bombs to achieve the same destruction capability. This lack of explosive power is offset by the size factor, however, since so many more bombs can fit into the bomb bay, so architecture #451 and all of the other architectures chosen have very good sorties utility.

Casing weight is a complex number that includes the guidance system, the metal that encases the explosive, and the bomb's wings. This number is very small for architecture #451, but as can be seen in the table, increases for rows #2, 3, 4, and 5 as the radius and the wing ratio increase. This also increases the cost, as much of the cost is determined by weight.

Finally, we have the bomb's main chosen dimension, diameter. Architecture #451 has a diameter of 5 inches, and through this diameter and the total bomb weight, the length of the bomb is determined. In our possible choice of point design, the total length is calculated to be 11.5 feet long, meaning that only one bomb can fit lengthwise into the bomb bay, which is assumed to be 20 feet long. All in all, the underlying model tells us that 20 of these can fit into the bomb bays of the F/A-22, which is why only three sorties are needed to destroy our chosen target set.

Of course, the calculated length cannot be more than the length of the bomb bay, and some architectures do not show up in the tradespace because they ended up being too long. One of these architectures was an attempt at having the following design vector values: 3" diameter, 220 pounds of explosive, and an aspect ratio of 2. This architecture's (architecture 8730) calculated length is 64 feet long, so it does not fit into the F/A-22's bomb bay, so it fails its

mission of having a small number of sorties to destroy the target set, so its point does not appear on the graph.

As expected from our earlier analysis, the main gain in utility for the pareto-frontier architectures is due to an increase in glide distance, since having a low number of sorties is a prerequisite for being on the frontier. These particular architectures will be tracked through spiral two. There is a one-to-one correspondence between spirals one and two; i.e., each architecture in spiral one has a single corresponding architecture in spiral two. In fact, they have the same architecture numbers.

These architectures will be tracked into spiral two, where there is a one-to-one correspondence. They will be tracked into the third spiral as well, but there is a one-to-four ratio there; i.e., each architecture in spirals one and two have four corresponding architectures in spiral three. This is due to the addition of four different fuel weights into the spiral three Design Vector for each architecture that exists in spirals two and one.

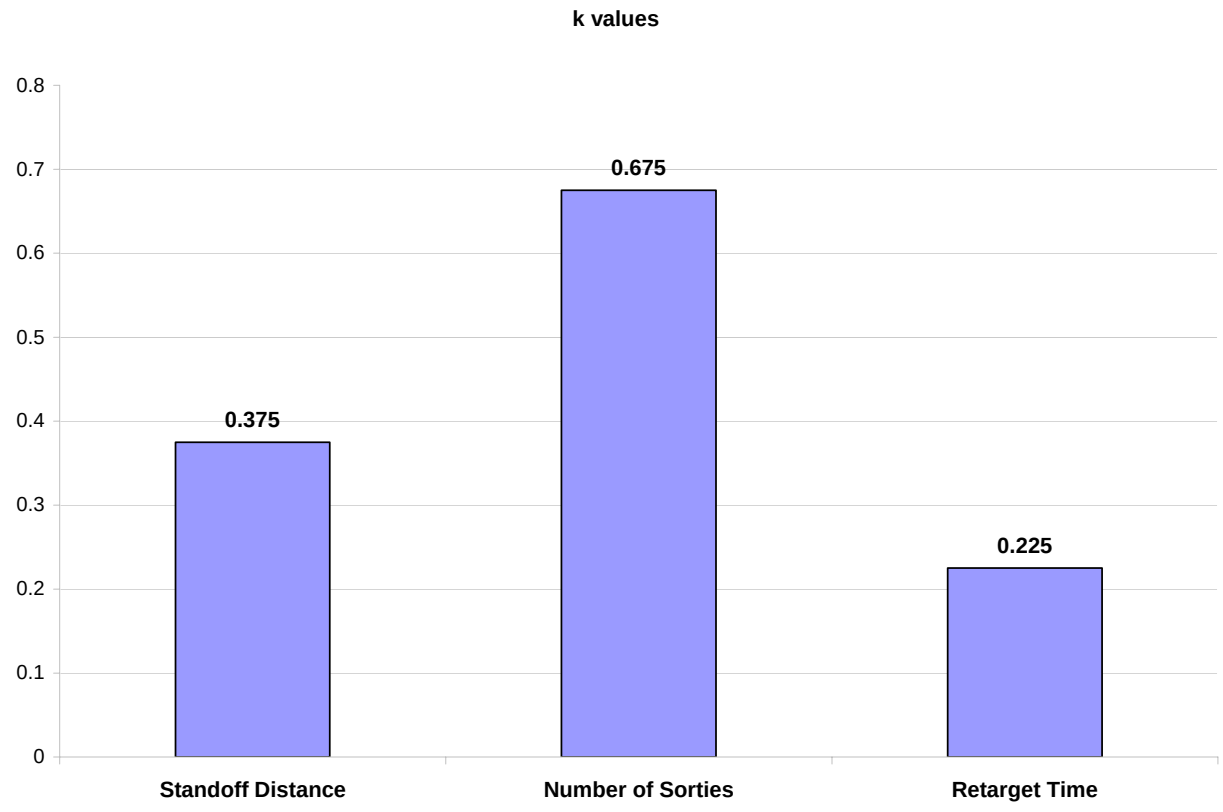
## 5.2 Second Spiral

As discussed in section 4.5, there is very little change to the physical bomb for the second spiral. A delta was added to the cost model, but as it was added to every bomb architecture, the relative cost should not be expected to change. However, the utility values will change due to the addition of a new attribute: retarget time.

Because we have not changed the model of the physical bomb itself, it would be logical to think that the same bombs would have similar utilities—i.e., the five architectures that we chose because they were at interesting spots on the pareto-optimal frontier would still be on the pareto-optimal frontier

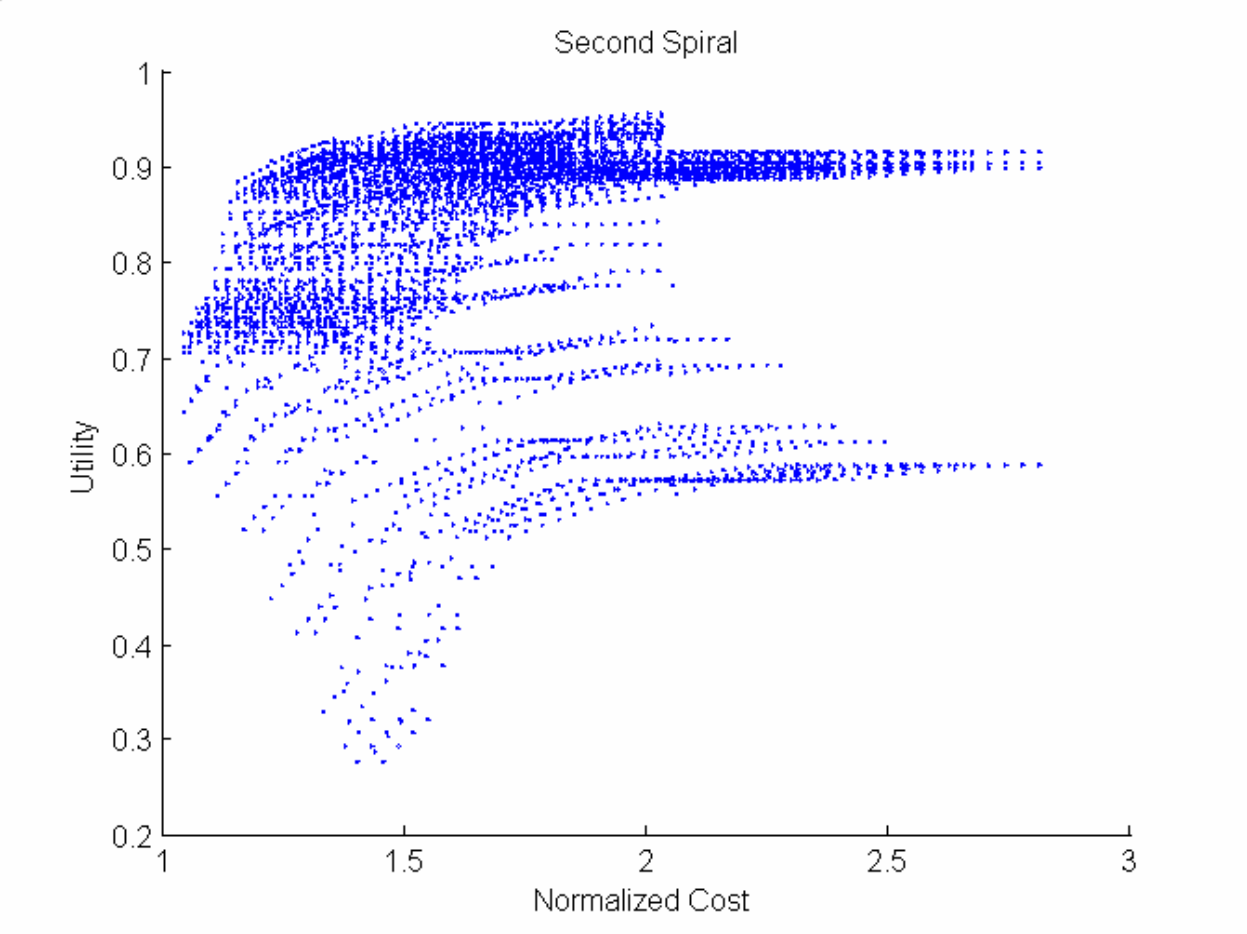
### 5.2.1 **k-Values**

The k-values (weighting values) are quite different for the second spiral. This is due to the user's consideration of a third attribute during the interviews. The k-values are:

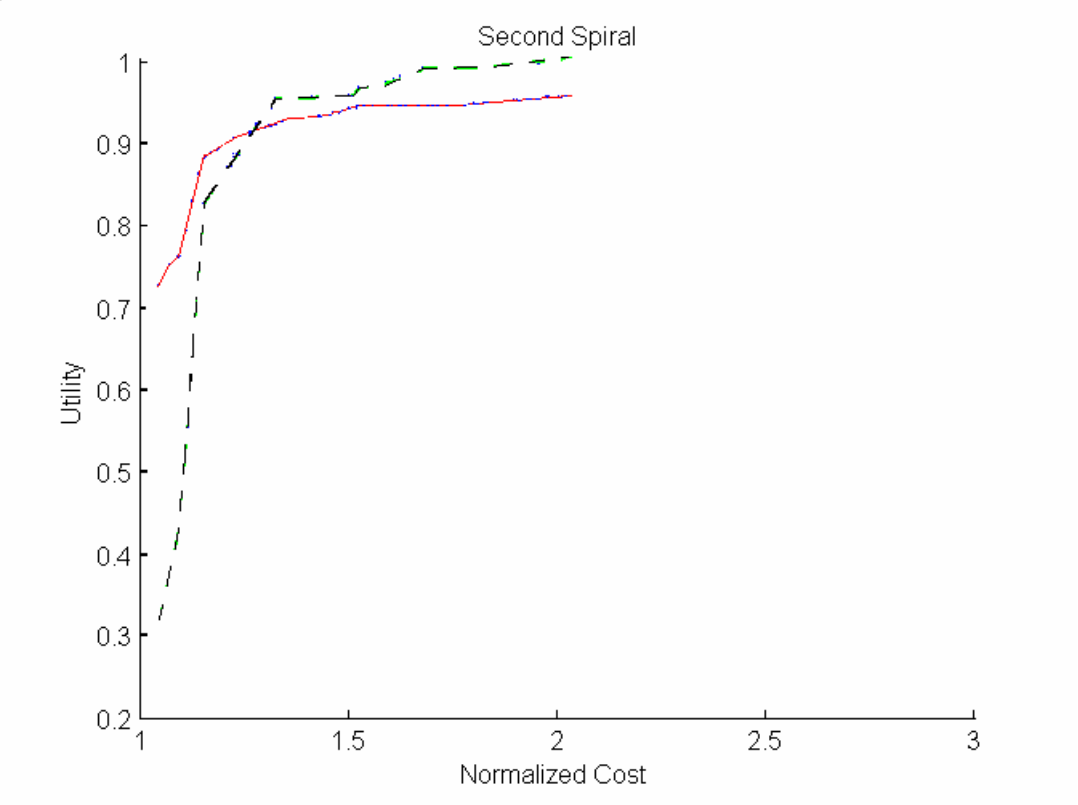



### 5.2.2 **Overall Tradespace**

These k-values reflect themselves in the tradespace by changing the weighting factors on each attribute. The second spiral has a tradespace that is quite different in details, but similar in shape to the first spiral:




The shape of the two different pareto-optimal frontiers, however, is quite different. They are presented here without the architectures for clarity. The dark green dotted line is the first spiral's pareto-optimal frontier; the red solid line is the second spiral's frontier.



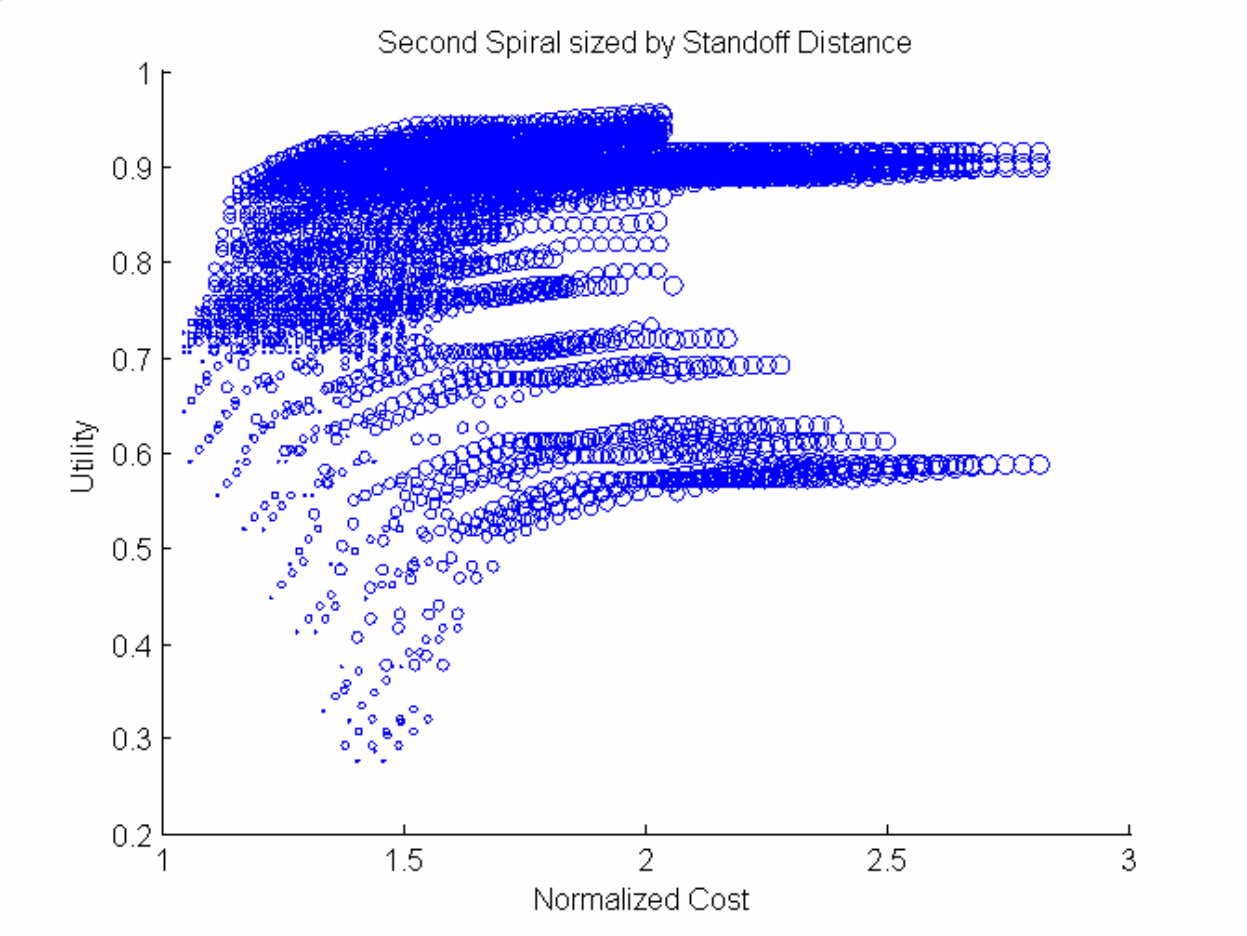

It is important to remember that the shape of these curves is not analytically comparable; they are presented here only to show that the knees in the curve are in different places and that the two curves are substantially different. What seems to be happening is that the less expensive architectures are at higher utility, but the more expensive architectures are at lower utility. This is due to the interaction of the various attributes in this particular model, especially the fact that the user shows that he is generally risk-averse in most of the attributes.

### 5.2.3 **Analysis by Attribute and/or Design Vector**

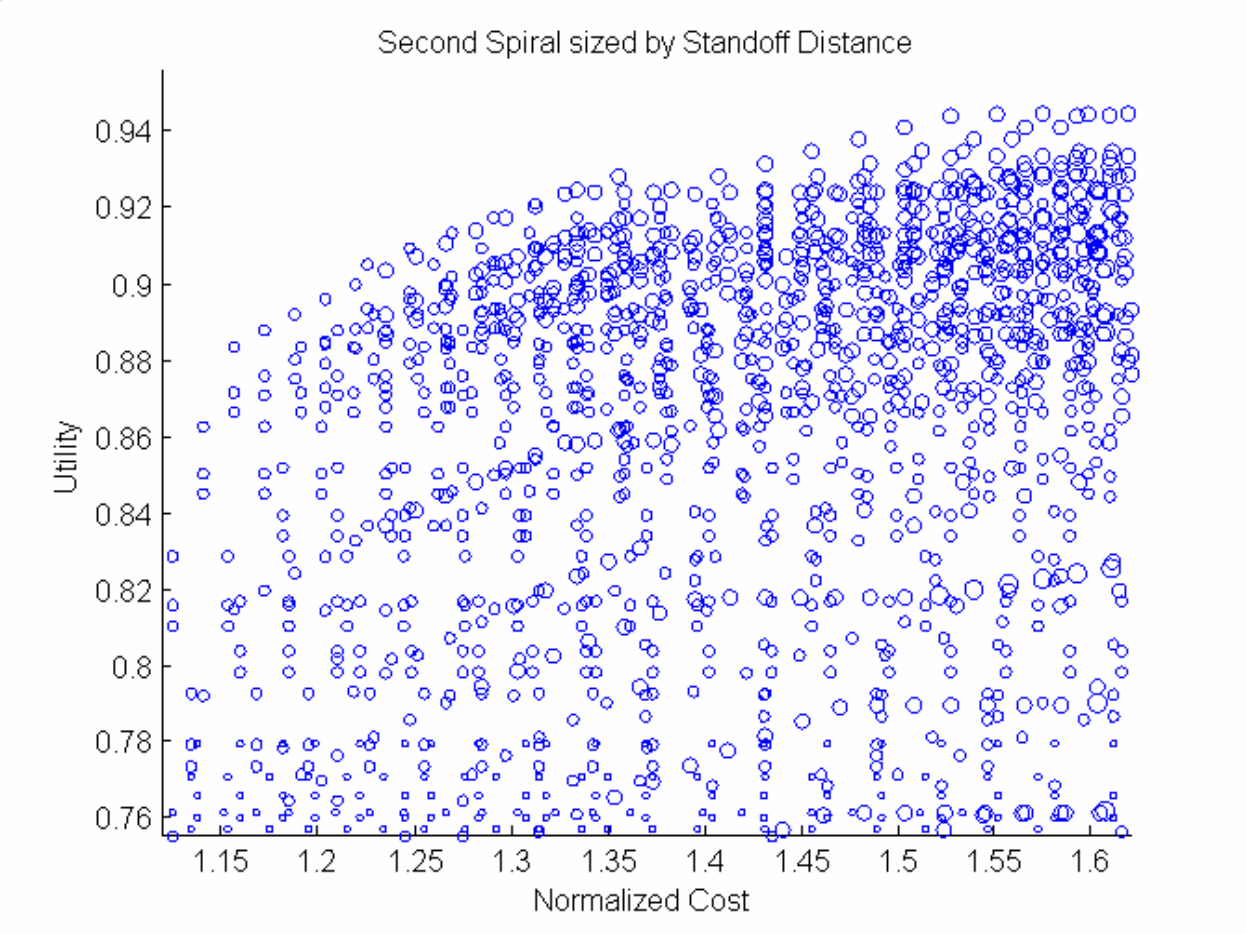
#### 5.2.3.1 Standoff Distance

A graph of the second spiral's tradespace with the circles sized by standoff distance is presented here. As a longer standoff distance is desired, a larger circle is better. The maximum and

minimum values are unchanged, with the largest circles at approximately 60 nmi and the smallest circles at approximately 20 nmi.



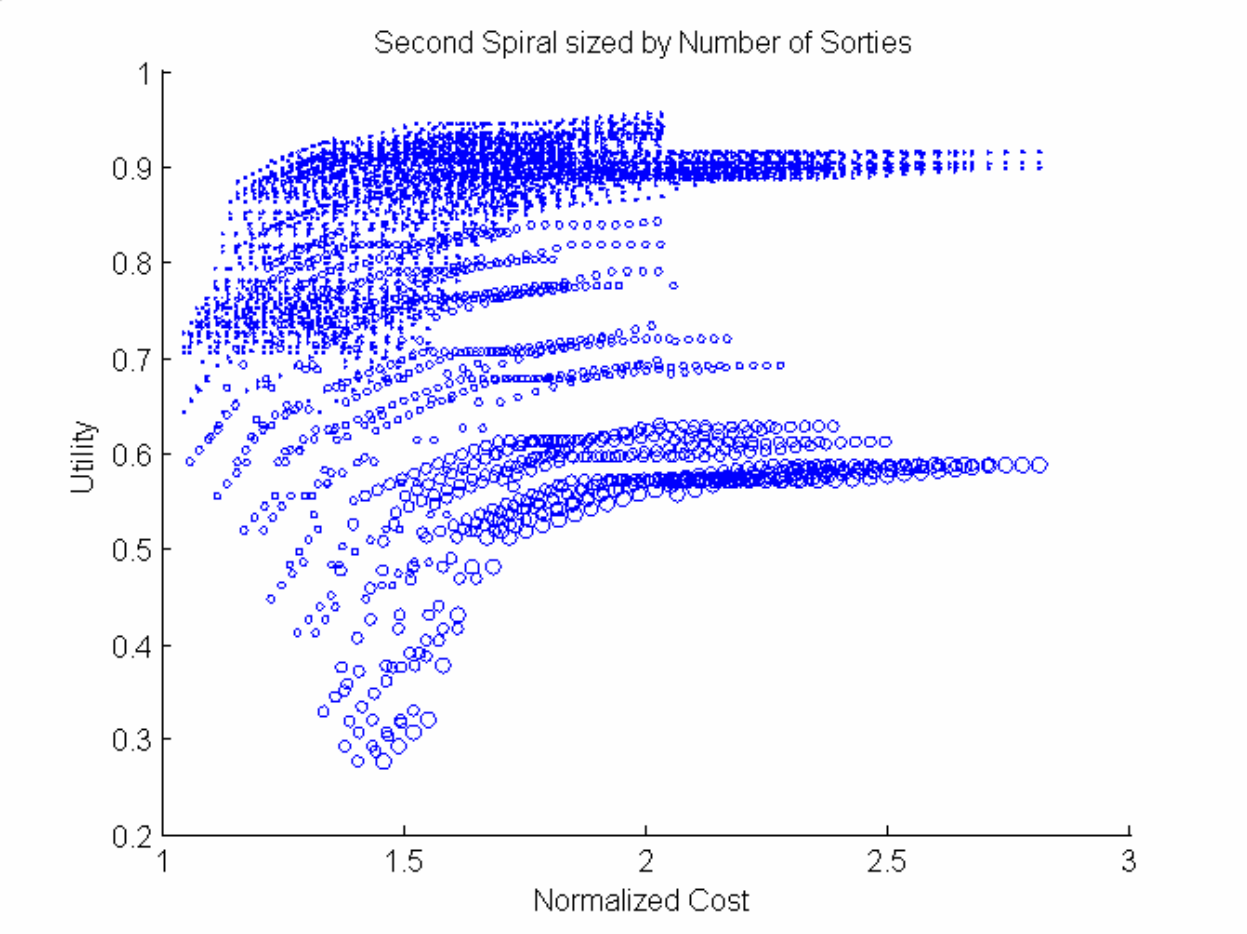

This graph shows that it is difficult, but possible, to achieve a higher utility with a lower glide distance. If we zoom in to the upper-left hand corner of the data, we see that standoff distance is varying quite a bit:




We can see in the various strata that the smallest glide distances start at the lower left of this zoomed graph and move up in utility only slightly for higher cost. It seems that the picture here is much more complex than in the first spiral, but standoff distance is still of great importance to the user.

#### 5.2.3.2 Sorties Required to Destroy Target Set

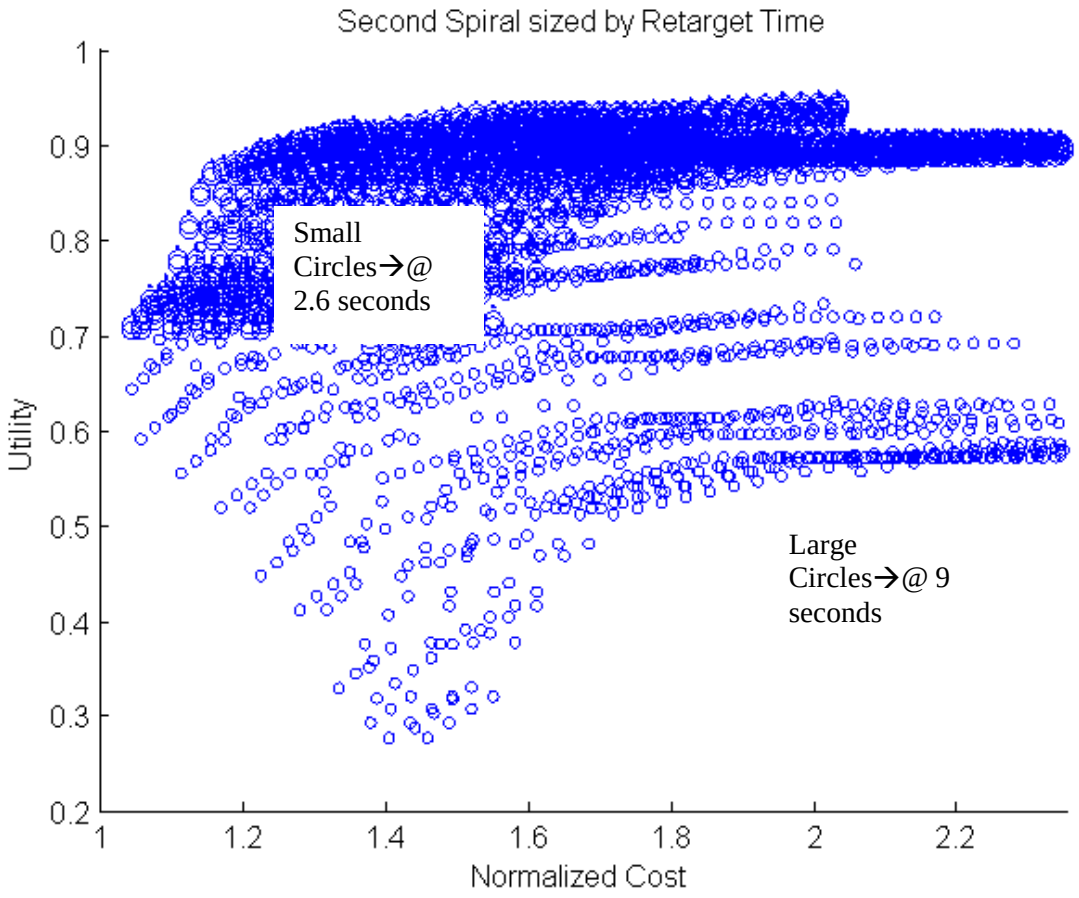
A graph of the second spiral's tradespace with the circles sized by sorties is presented here. As a smaller number of sorties is desired, a smaller circle is better. The limits on this tradespace are the same as in the first spiral, with the largest circles being 40 or more sorties, the smallest circles being 2 sorties.



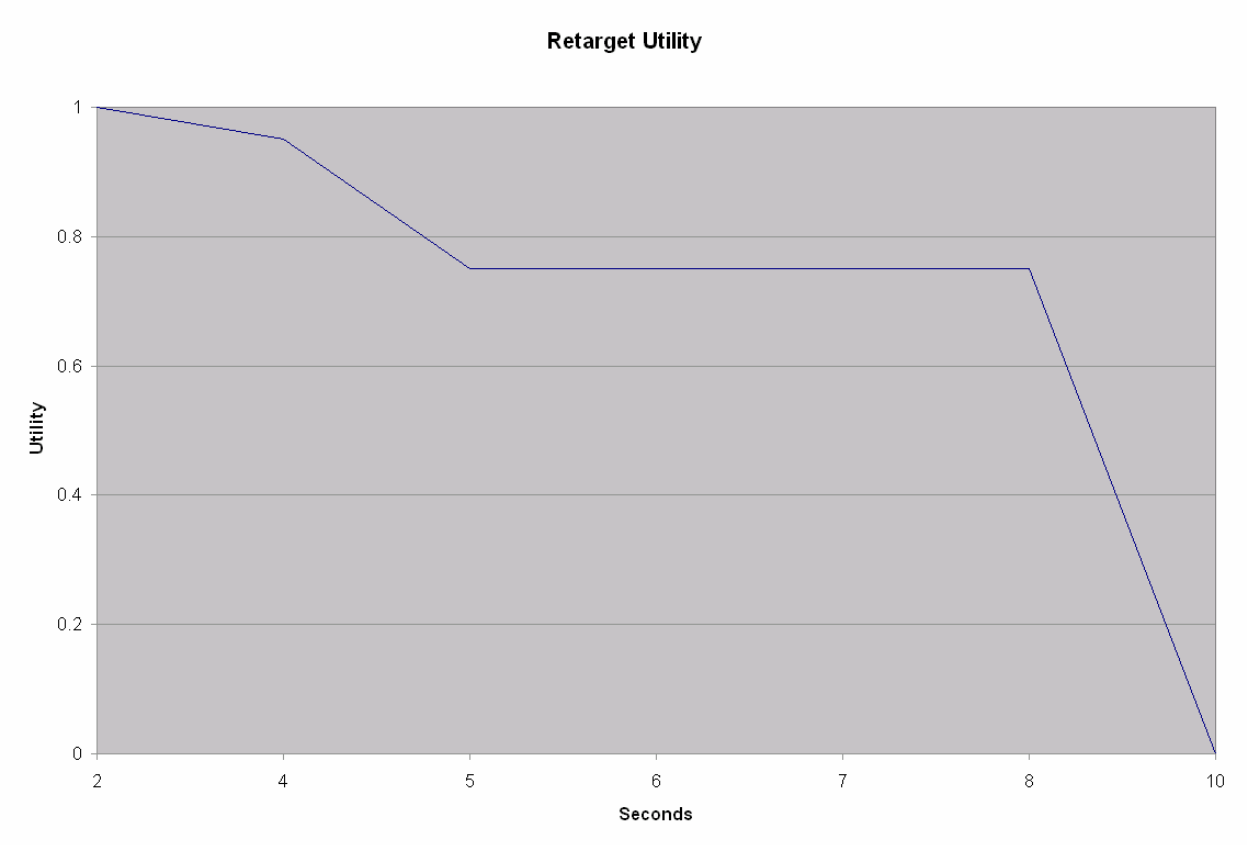

It is quite clear from this graph that it is once again necessary to have a small number of sorties to reach high utility.

#### 5.2.3.3 Retarget Time

A graph of the second spiral's tradespace with the circles sized by retarget time is presented here. A smaller retarget time is desired, as this gives more flexibility to retarget just before the bomb impacts, so a smaller circle is better. The largest circles on this graph represent a retarget time of 9 seconds, the smallest circles represent a retarget time of 2.6 seconds.




This graph is very scattered. There are small circles along the pareto-optimal front, but there are large circles right below them. Clearly, retarget time is a secondary driver for the second spiral. This is explained by the single attribute utility results generated and presented in chapter 3, the graph is reprinted here for the reader's convenience:




As can be seen by this graph, the user values retarget time below 8 seconds. The ability to retarget is useful, but, as shown in the graphs in section 4.5.3.3, the equipment available today is so good that a bomb can be accurately retargeted to a target up to 20 degrees out of line within 5 seconds of impact. Thus, the retargeting utilities are high for almost every architecture.

However, it should be remembered that, while the difference in utility between the points is low, **the amount of difference between utilities means nothing**. One can only draw the conclusion that the user receives more utility from an architecture that has a smaller retarget time, one cannot make any conclusions as to how much this affects the utility of the system.

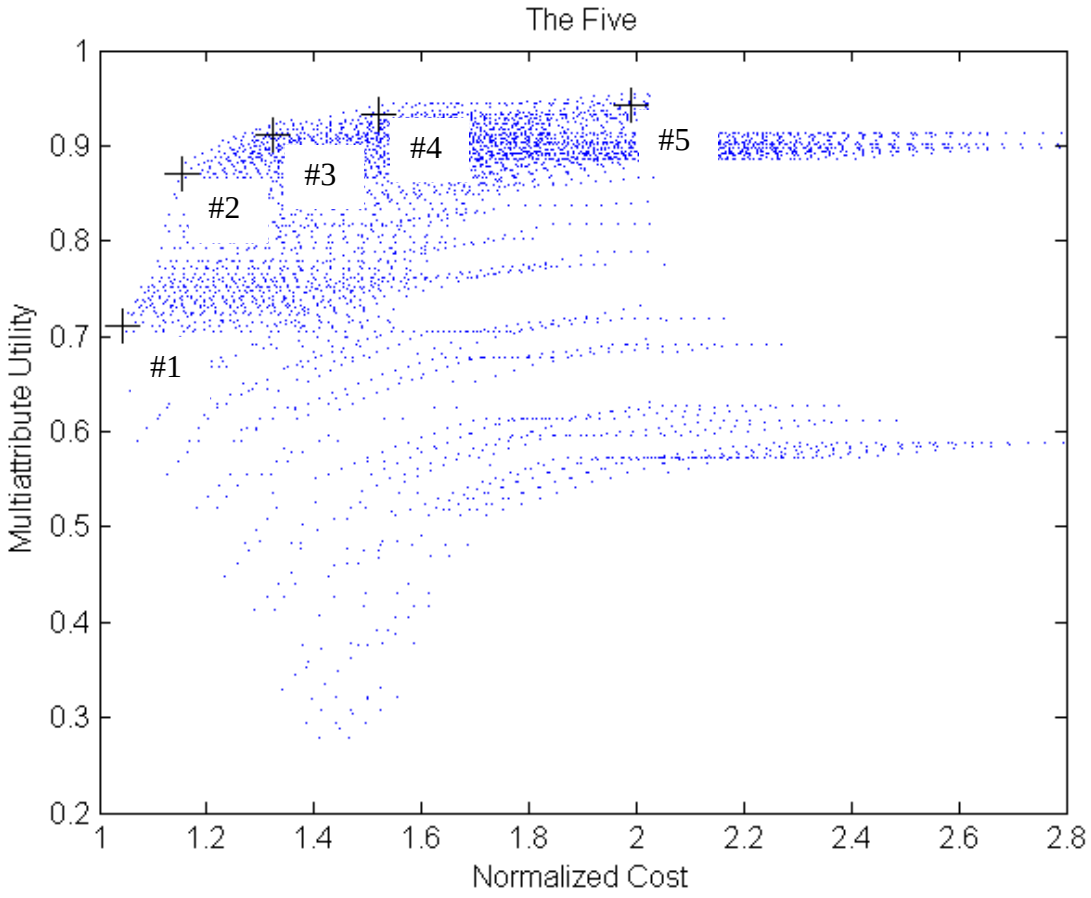
In fact, the fact that there are smaller retarget times above slightly larger retarget times shows us that the user does indeed value more highly the architectures with a more flexible retarget time.

What is interesting is that the k-value table in section 5.2.1 shows us that, while the k-value for retarget time is smaller than the k-value for standoff distance, it is of the same general size. We have seen from our analysis that standoff distance is a major driver whereas retarget time, the reason for the second spiral, is not a driver. This is due to the k-value, but also due to the fact that retarget time varies only a small amount—thus, the small amount of variation given by the physical model is made into a smaller utility variation from the k-value.

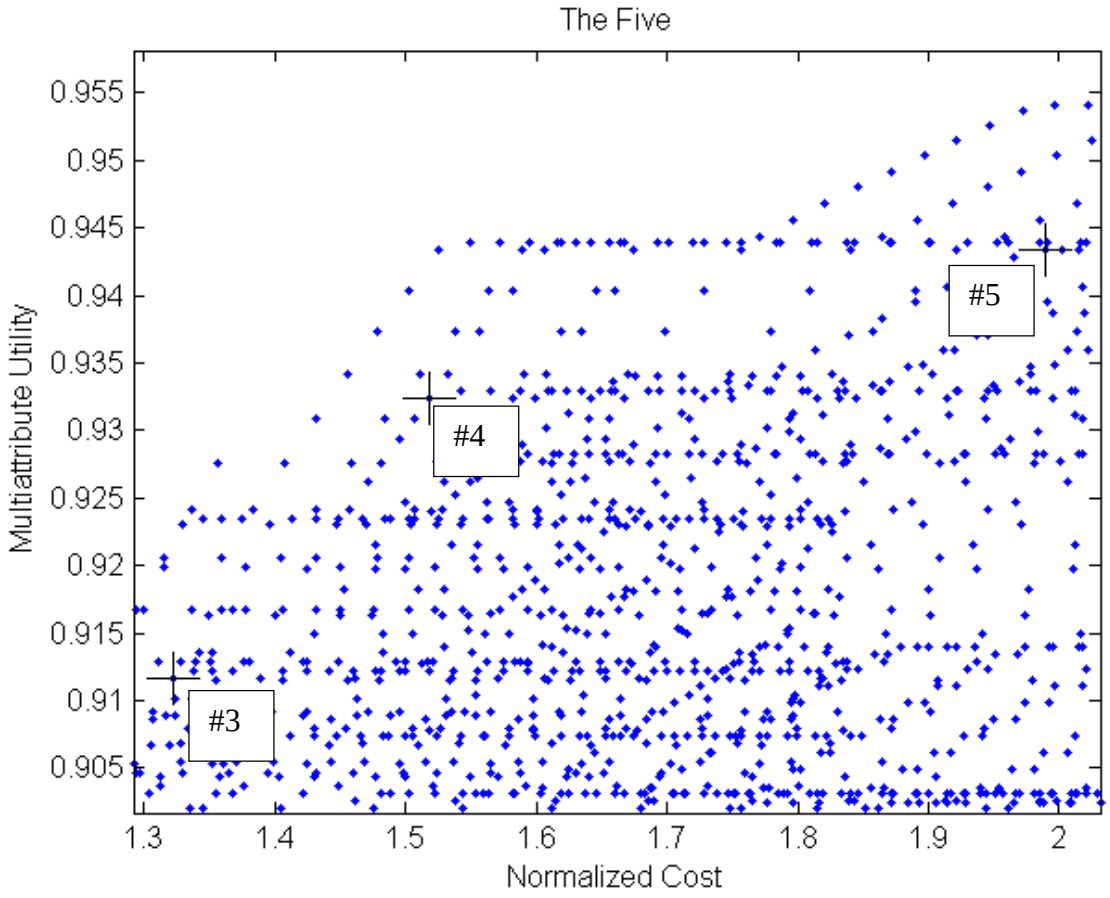
The interplay of these two kinds of variation within the product function for multiattribute utility can give the engineer pause. This shows why Keeney & Raiffa were quite specific that the k-values are not direct weighting values (Keeney & Raiffa, section 6) but rather they are indirect weighting values. If they were direct weighting values, we could directly compare various utilities on our graphs. Careful analysis of the whole tradespace is needed to understand fully what is going on.

### 5.2.4 **Analysis by Specific Architectures**

We now return to our five architectures and see how they look in the new tradespace. First, a graph of where they are now:




It is can be seen in this graph that several of the architectures are no longer on the pareto-optimal frontier. This is shown more clearly in the close-up on the three outclassed architectures, below:




It is clear from this graph that these architectures are sub optimal, that is to say, it is possible to achieve a higher utility for less cost.

Here are the previous charts, with Retarget time and utility added, and the changed utilities in bold. First, the utility chart is shown, after which the design vector chart is shown. Note that nothing actually changes in the design vector chart—as stated above, the model of the physical bomb is identical in this spiral.

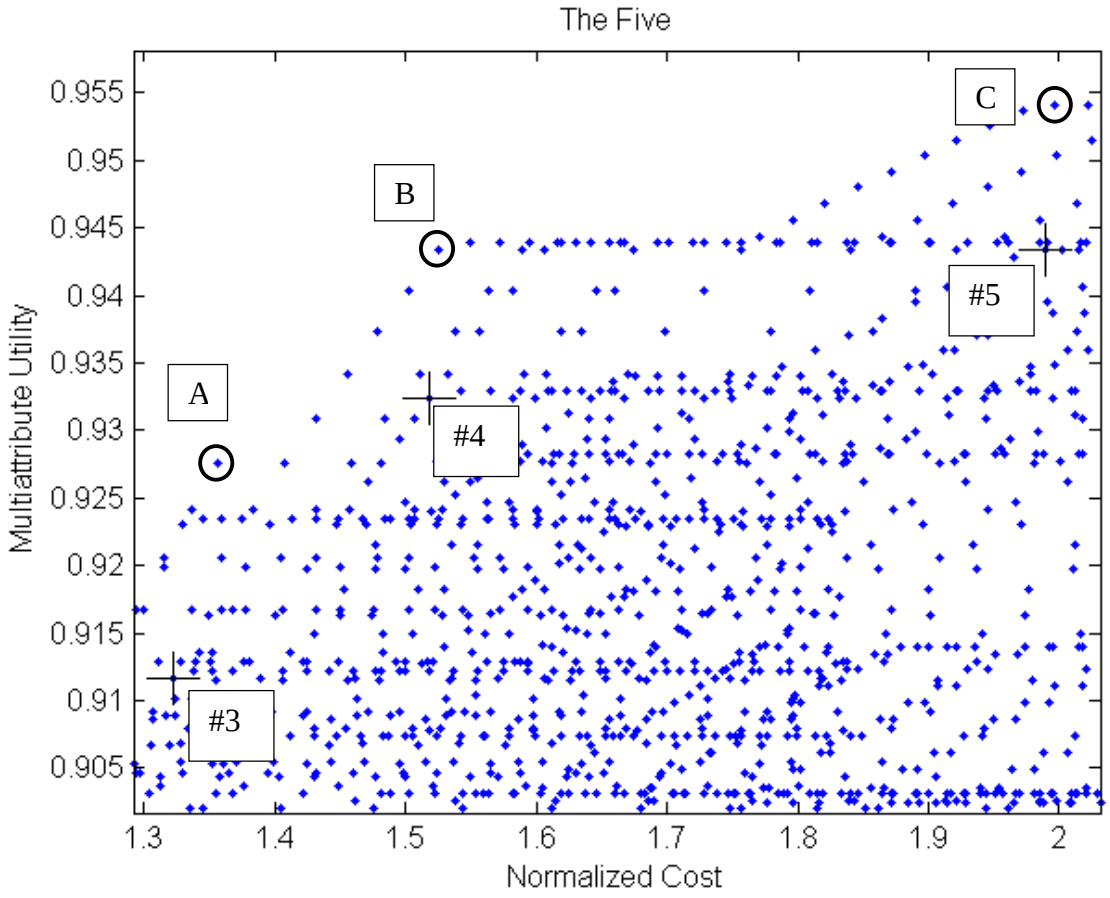
| | # | Utility | Cost (Normal-ized) | Standoff Distance (nmi) | Standoff Single Attribute Utility | Sorties Required to Destroy Target set | Sorties Single Attribute Utility | Re-target time | Re-target utility |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2,577 | **0.7109** | 1.0446 | 20.52 | 0.0111 | 3 | 0.9583 | 4 | 0.4500 |
| 2 | 2,580 | **0.8712** | 1.1545 | 33.62 | 0.7588 | 3 | 0.9583 | 2.6 | 0.5550 |
| 3 | 2,505 | **0.9117** | 1.3226 | 46.83 | 0.9476 | 3 | 0.9583 | 4.0 | 0.4500 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 2,595 | **0.9324** | 1.5184 | 46.83 | 0.9476 | 2 | 1 | 2.6 | 0.5550 |
| 5 | 3,859 | **0.9433** | 1.9896 | 60.42 | 1 | 2 | 1 | 2.6 | 0.5550 |

| | Arch. # | Aspect Ratio | Guidance Type | Number of Actuators | Explosive Weight | Casing Weight | Diameter |
|---|---|---|---|---|---|---|---|
| 1 | 451 | 0.20 | Inertial only | 2 | 150 | 12.5 | 5 |
| 2 | 2019 | 0.55 | Inertial only | 2 | 150 | 34.4 | 5 |
| 3 | 4483 | 1.10 | Inertial only | 2 | 150 | 68.8 | 5 |
| 4 | 4486 | 1.10 | Inertial only | 2 | 150 | 110.0 | 8 |
| 5 | 8071 | 1.90 | Inertial only | 2 | 150 | 213.8 | 9 |

At this point, the five architectures are still in the same rank order of utility, but the overall utility numbers have changed. Points #4 and #5 are both lower in utility than they were in the first spiral's tradespace, but points #1, #2, and #3 are all higher in utility. The addition of the new attribute has changed the tradespace significantly, and what was once optimal is no longer so. Points #3, 4, and 5 are all sub optimal even though they were optimal on the first spiral.

However, there are three architectures that we should look at, shown as points A, B, and C in the graph below:




These points are the new pareto-optimal frontier architectures that are above the outclassed architectures from the first spiral. They are, respectively, architectures 3421, 4542, and 8127. These three architectures have one thing in common that is different in their design from all of the previous architectures considered: they have three actuators instead of two. This is because the fundamental base for retarget time is accuracy, and accuracy is based on guidance type and number of actuators.

The reason why GPS never makes it to the top is because of the way that retarget time is defined: the time before impact that the bomb can be retargeted, with the requirement of **having the same accuracy as before retargeting.** Since the bombs all use the same equipment for steering but each different type of guidance system has a different expected accuracy, it takes the

bombs longer to retarget for systems that expect higher accuracy. This accuracy is much tighter in the case of architectures with GPS systems, and thus the retargeting time is higher for them than it is for the inertial only guidance systems.

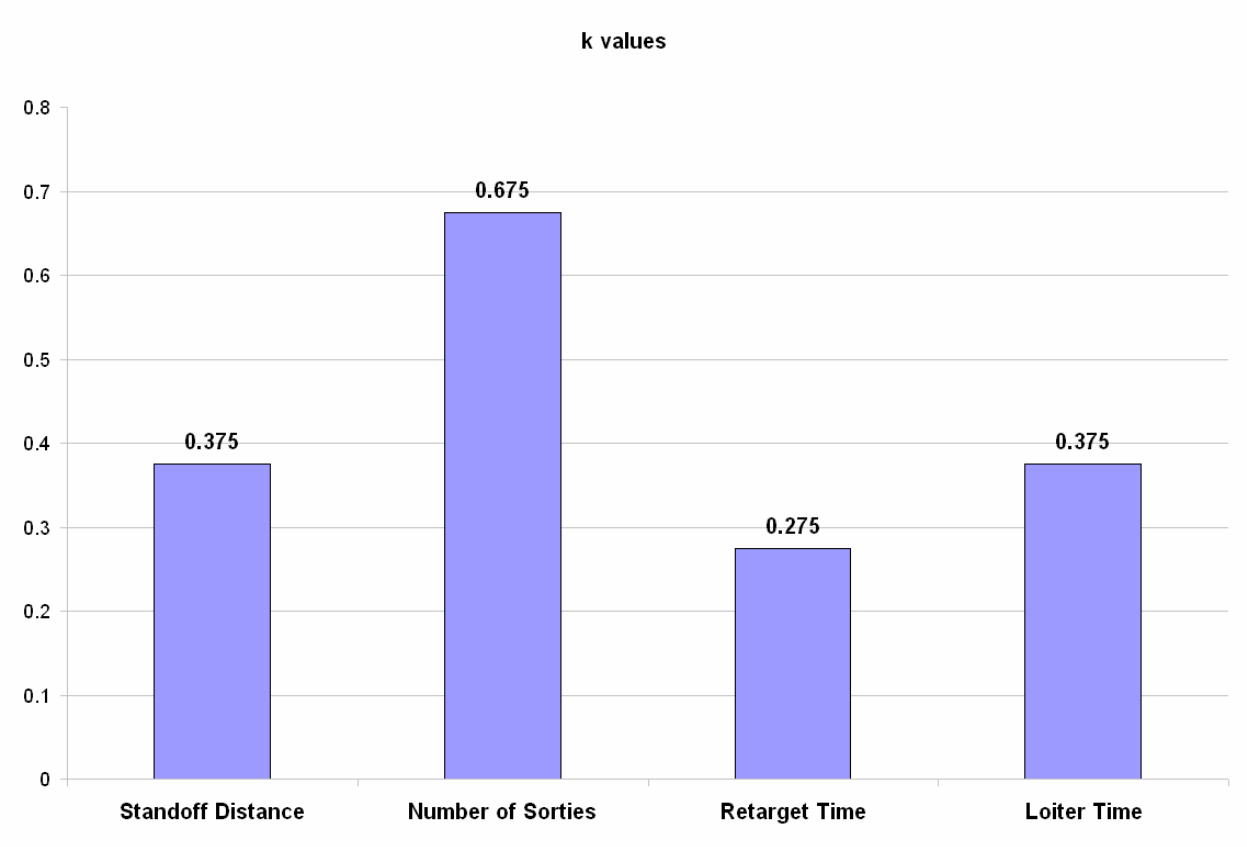
Again, the true underlying physical bombs that are being modeled here are nearly identical to the first spiral—only the addition of software that allows retargeting and a receiver have been added. It is interesting and appropriate that MATE models the addition of a new mission for the bomb in such a way that the bombs' utility changes despite the fact that the bomb itself remains unchanged. Clearly, the MATE system is capturing the changing user utility for the changing system.

## 5.3 Third Spiral

This spiral adds Loiter Time to the model. Loiter time is the time in minutes that a bomb can loiter over its target zone once it has gotten to that zone. Both Standoff Distance and Loiter Time are useful for the Air Force: standoff time protects both planes and human life by allowing a pilot to launch the bomb a number of nautical miles away from the target; loiter time allows the Air Force the ability of area control by maintaining a group of bombs over such an area. It is possible for these bombs to be launched from a plane dozens of nautical miles away, and then these bombs can control a vital pass or bridge by threatening to drop on any military target that attempts to cross underneath them.

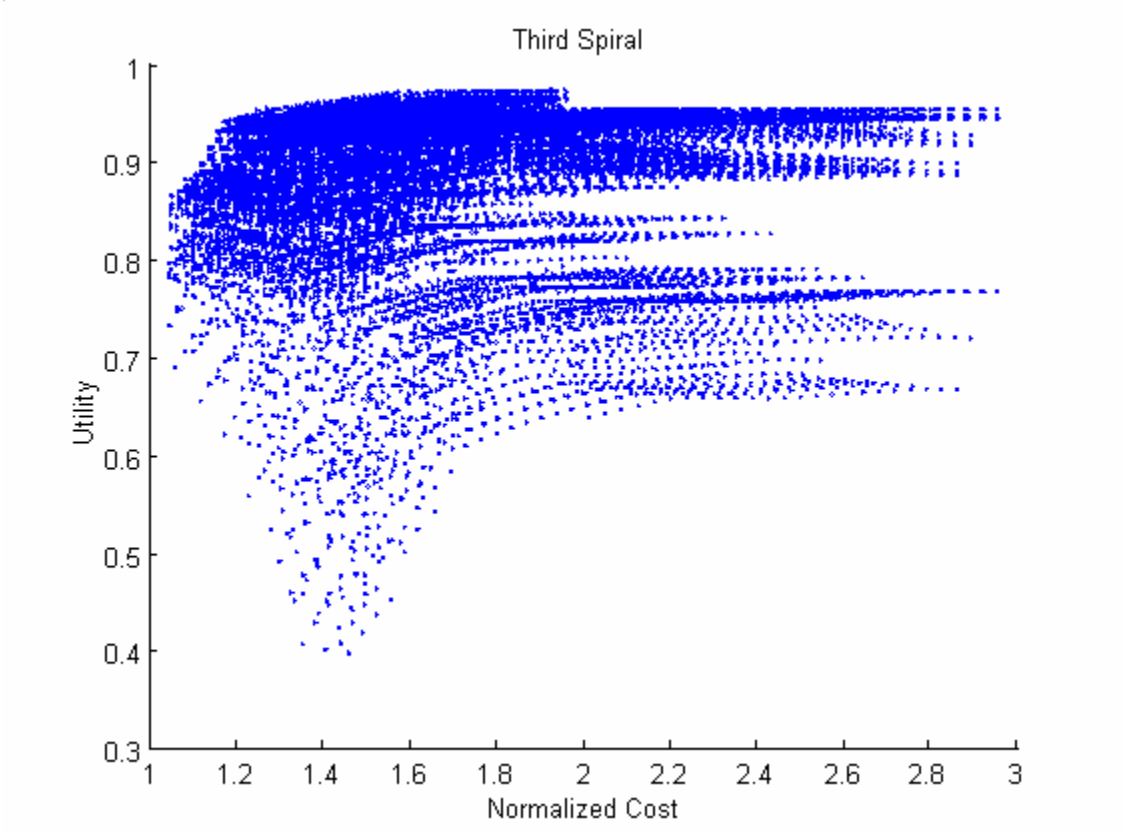
### 5.3.1 **k-Values**

The k-values of the third spiral are only slightly different than last time. Retarget time has increased slightly from 0.225 to 0.275. The new attribute, Loiter Time, has a relative value equal to that of standoff distance, 0.375:



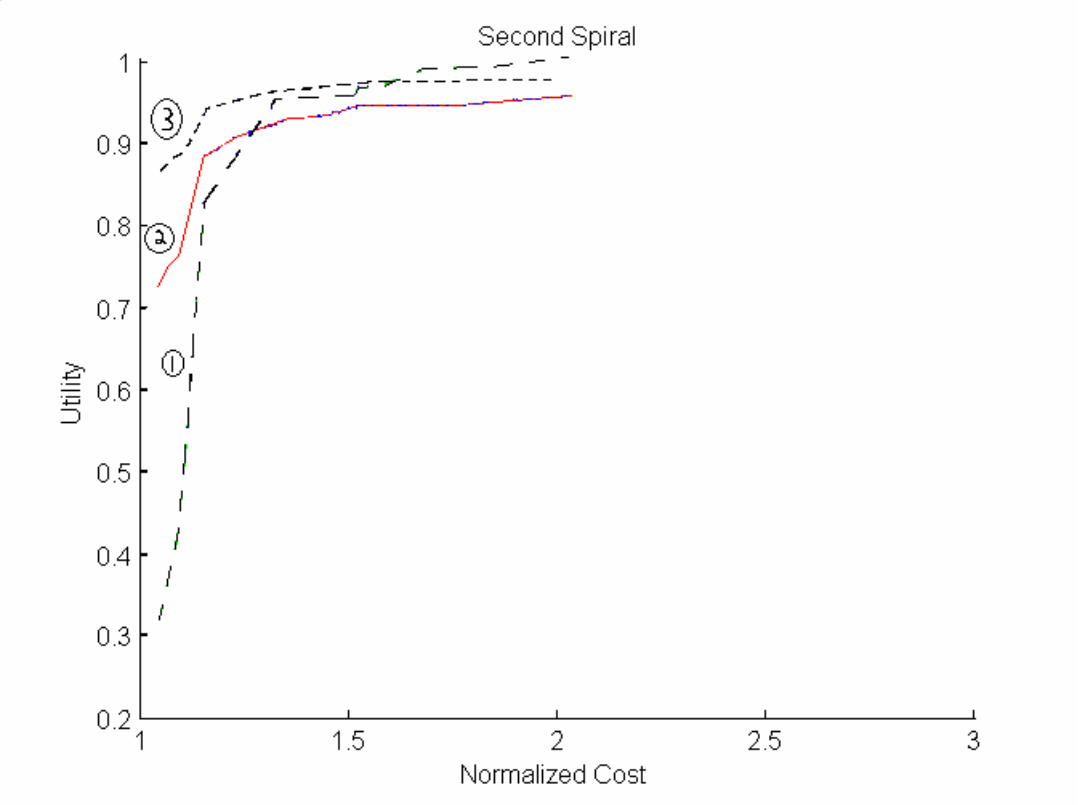

### 5.3.2 **Overall Tradespace**

Again, the tradespace is similar in its general shape, but quite different in details.




One difference can be seen immediately in the shape of this tradespace, though, and that is that it contains many more architectures. This is due to the addition of fuel weight into the Design Vector. Fuel weight has four different possibilities and thus multiplies the total number of possible architectures by four; however, of the 34,944 architectures, over 13,000—over one third of the total architectures—fall out of the tradespace due to the increased size of the bomb to incorporate the fuel tank. This increased size makes many of the architectures too long to efficiently fit into the F/A-22's bomb bay, and causes the architecture to fail its mission, specifically by increasing the number of sorties to destroy a target set up over 40. In the end, though, there are nearly three times as many architectures as in the first and second spirals.

The pareto-optimal frontier has changed again. Here it is in comparison to the previous two:



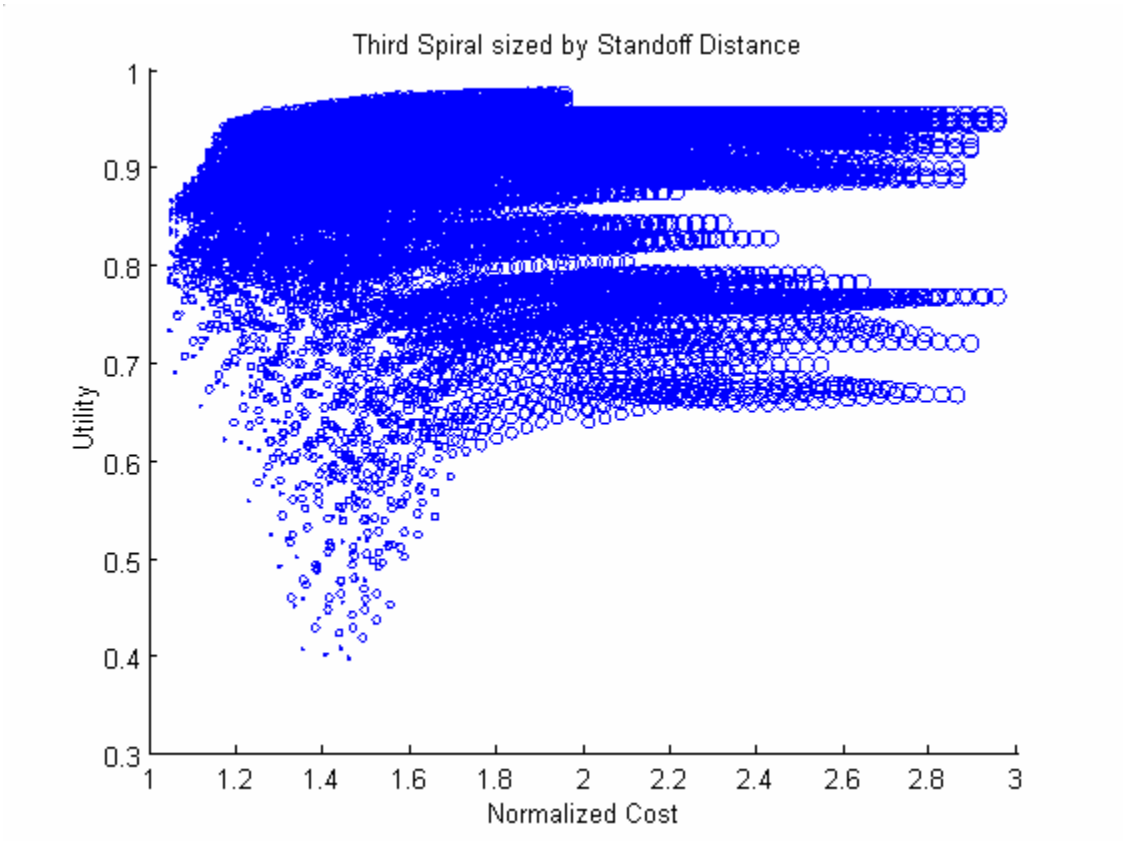

Again, it should be noted here that these pareto frontiers are not analytically comparable; they are presented here to show that the frontier changes significantly for each spiral, a fact that is easy to see graphically. The curve again shifts up on the low cost architectures, but remains higher in cost throughout due to the increase in cost from the addition of the turbojet.

### 5.3.3 **Analysis by Attribute and/or Design Vector**

This tradespace takes a significantly longer time for the computer to process. In addition, the number of architectures makes sizing the dots on the graph more difficult; they tend to overlap and obscure each other. There will be a significantly larger number of graphs in this section of the paper due to this, as each graph will be shown once for overall structure and then the scale increased to show the finer substructure.

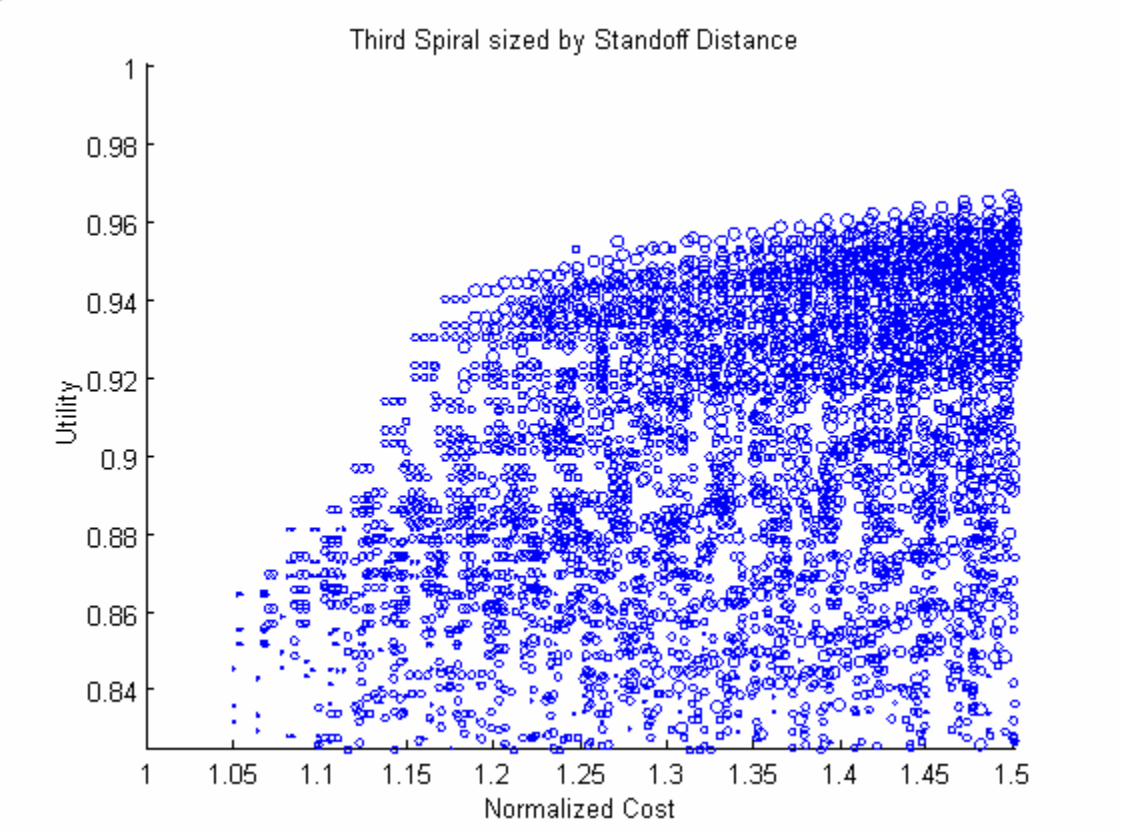
#### 5.3.3.1 Standoff Distance

The graph with the dots sized for Standoff Distance is printed below. The larger the circle is, the larger and more desirable the value for Standoff Distance, with limits again at 60 nmi for the largest and 20 nmi for the smallest.




Despite the confusion from overlapping points, an overall structure can be seen: the right-hand tails of the plot show only large circles, representing larger standoff distance; the left side of the plot shows only small standoff distances. In addition, the bottom of the graph seems to be populated with smaller standoff distances, whereas the upper portion is a mixture, at least on the left where separate dots can be discerned.

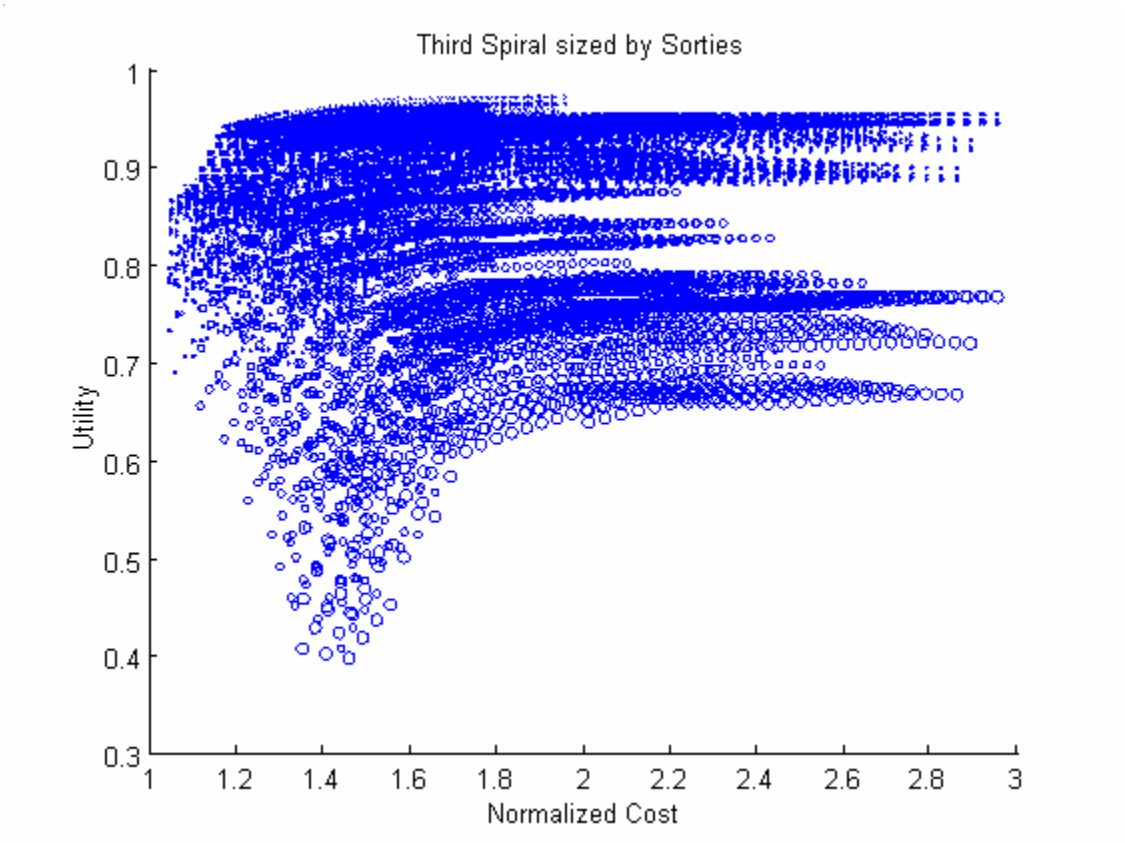
Zooming in to the upper-left of the plot, we see a bit more:



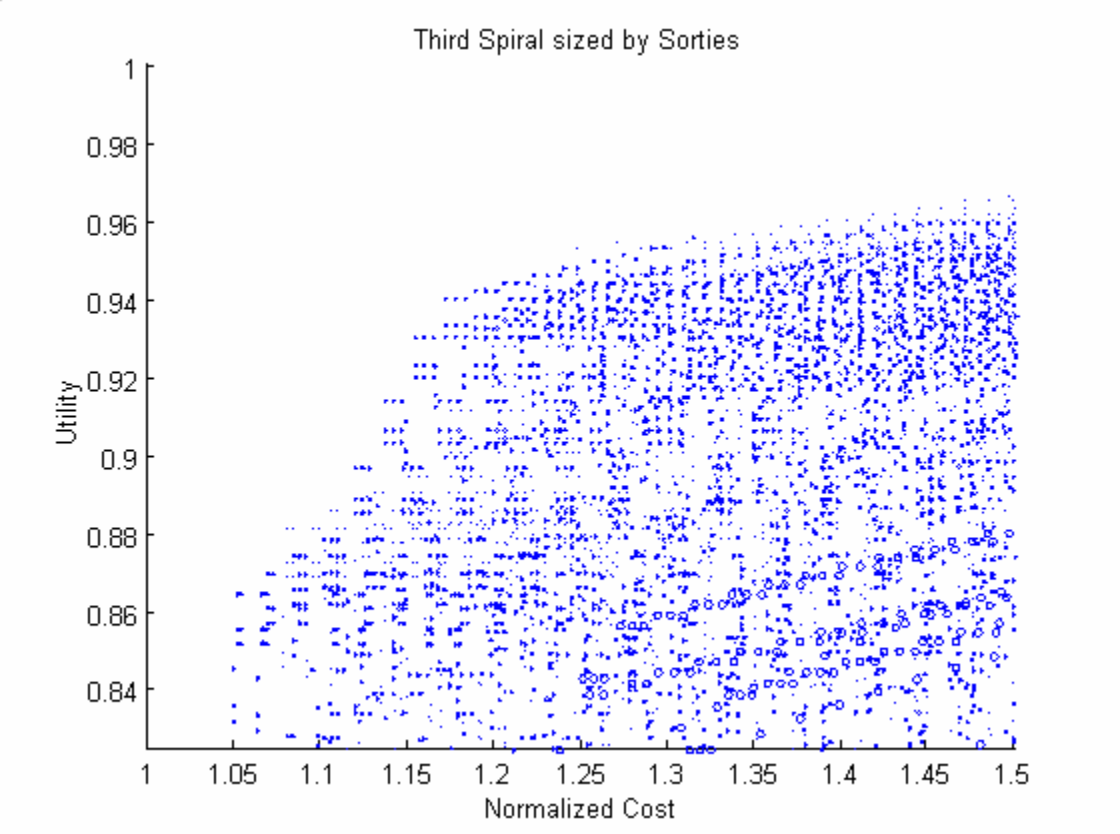

The mixture in this area is fairly thorough, with small standoff distance next to medium standoff distance. However, three distinct areas are discernable: the bottom 1/3 of the graph is composed of smaller standoff distance in pairs, triplets, and quadruplets. The middle 1/3 or so has more white space in it, and there are only small and very small standoff distance—none of the architectures with the smallest possible standoff distance make it into this region. Finally, the top 1/3 of the graph shows a denser mixture of medium to small standoff distance, but no smaller standoff distance circles can be seen. This indicates that a larger standoff distance (up to the maximum of 60 nmi) is necessary to have a higher utility; since the trend from bottom left to the top and right of the graph, it is clear that standoff distance is still one of the main system drivers during the third spiral.

#### 5.3.3.2 Sorties Required to Destroy Target Set

The tradespace of the third spiral sized by the number of sorties follows. The smaller the dots, the more desirable the value for the number of sorties, with the limits at 40 or more sorties for the largest circles, and 2 sorties for the smallest circles.



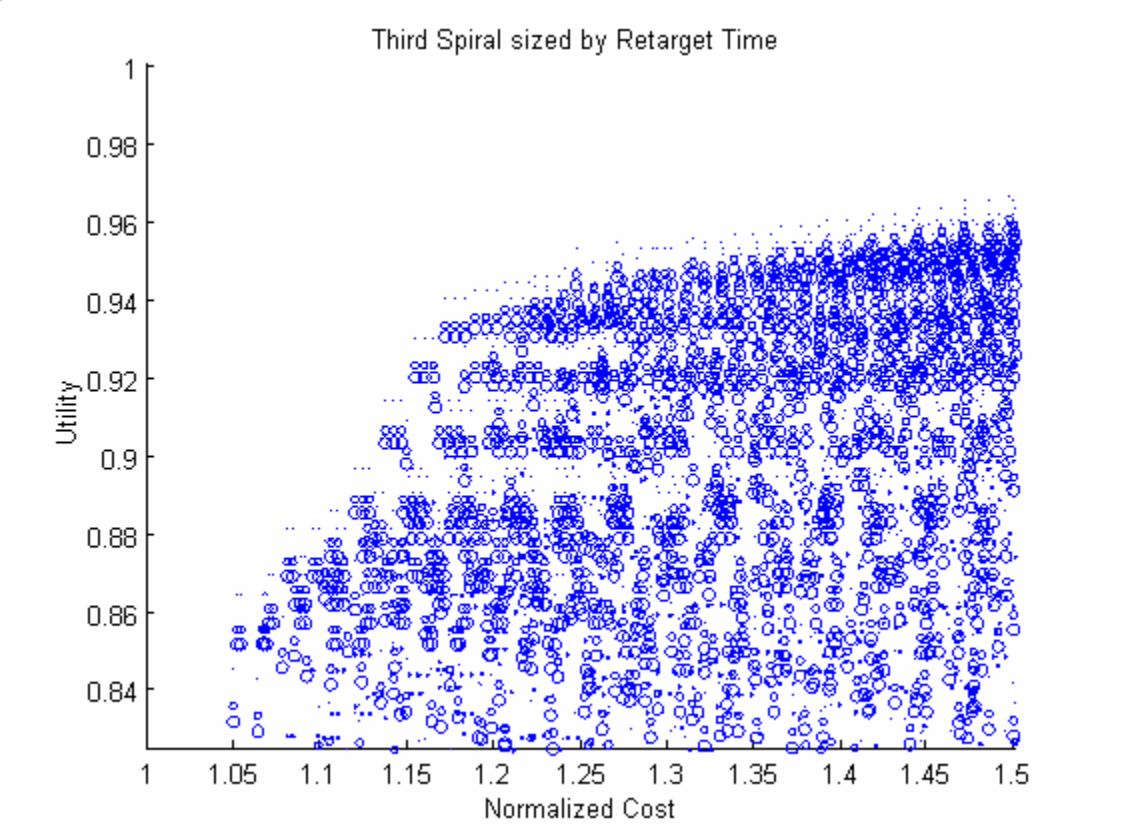

It is clear from the smallest dots being along the pareto-frontier and the larger dots on the bottom that, once again, utility is strongly correlated to number of sorties to destroy a target set. It appears from the enlargement below that it is again impossible to be on the pareto-optimal front unless the architecture has a minimum number of sorties:




The number of sorties here are minimum or very small, not just on the pareto-optimal frontier, but for a good distance into the tradespace as well. Even for the third spiral, the number of sorties to destroy target set is a primary driver. The best value, represented by the smallest dot on the graph,

#### 5.3.3.3 Retarget Time

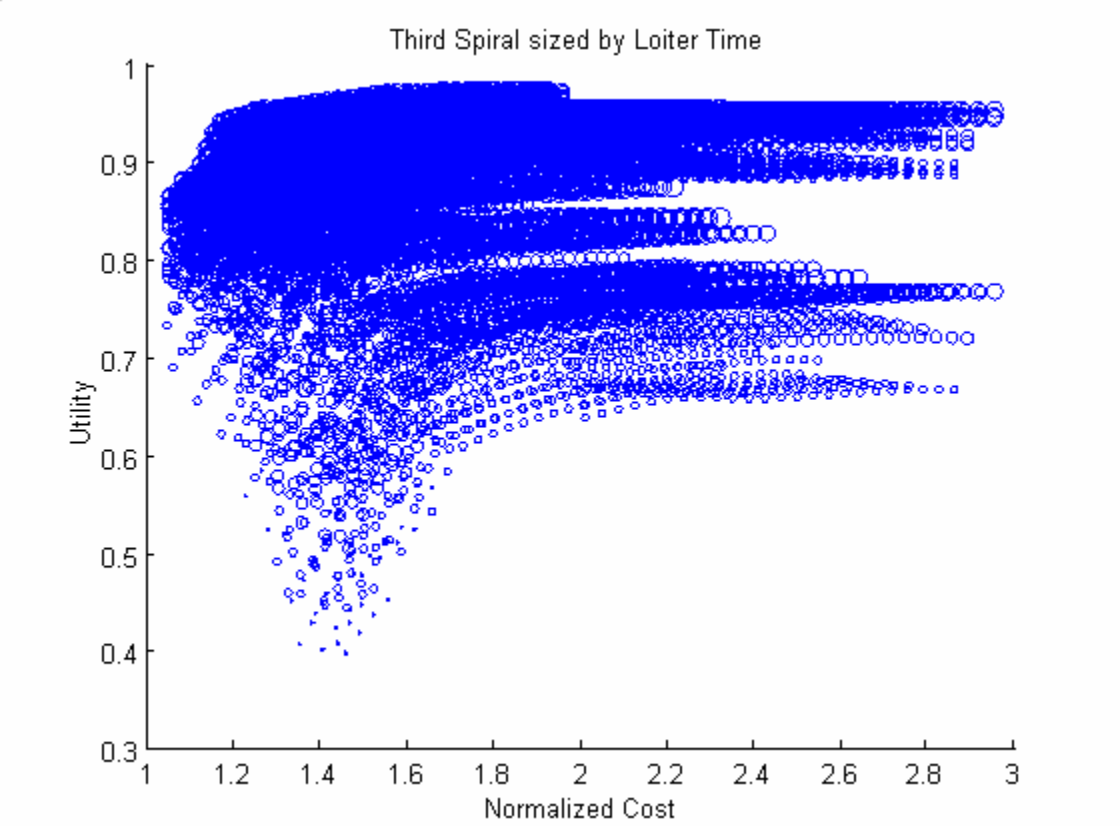
Since the tradespace for the third spiral with the circles sized by the relative retarget time of each is so confused as to be completely useless, presented here is the close-up of the upper-left corner of the graph, with the limits again being the largest circles representing a retarget time of approximately 9 seconds and the smallest being approximately 2.6 seconds.



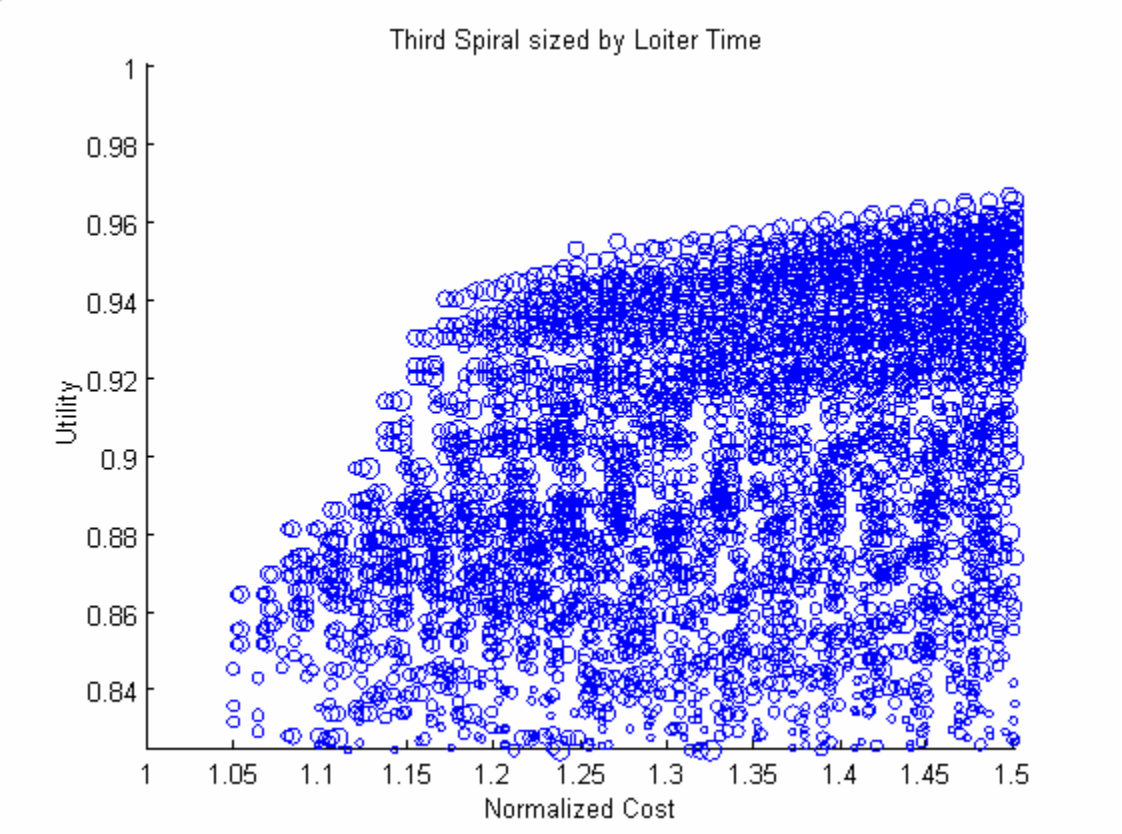

Here we see an interesting structure: there are clusters of three medium circles, with three smaller circles just above them, with three points a bit further up. Again, a smaller dot is more useful with retarget time, as it represents the ability to retarget closer to impact. This is an indication of this attribute being a secondary driver; the two primary drivers are determining the placement of each grouping of nine architectures, and retarget time is determining how the architectures are ranked within each group.

#### 5.3.3.4 Loiter Time

In the case of Loiter Time, the larger the number, and therefore the circle, is, the more utility given to the user. The largest circles on this graph represent loiter times of 120 or more minutes, well over the initial range set for the interview. The smallest circles represent a loiter time of approximately 11 minutes.



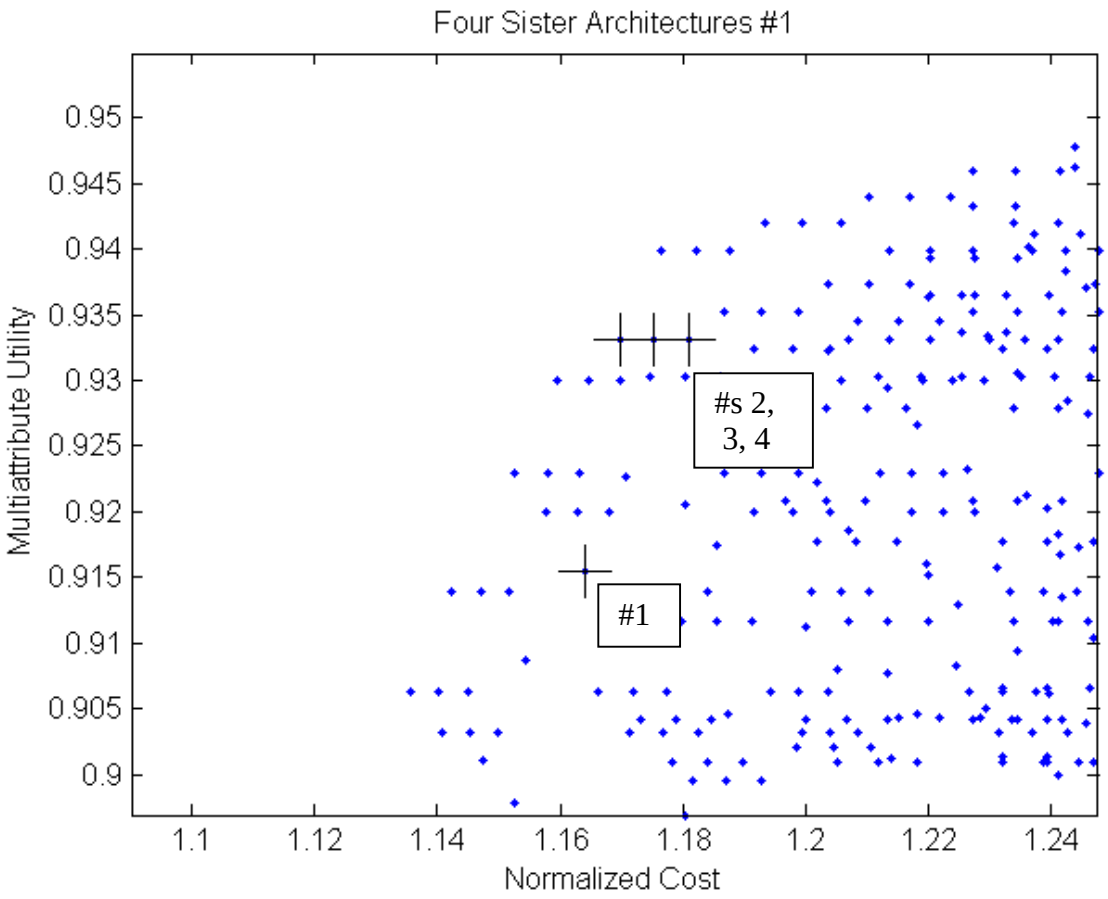

As can be seen, the bottom of this plot shows the smaller loiter times; the larger loiter times are in the top half. The enlargement of the pareto-frontier, though, shows a very interesting structure:




In this close-up of the plot, we see that there are several groups of utilities that have increasing loiter time but do **not** have increasing utility. In cases like this one, we might expect to see four architectures grouped together because we have four architectures in the third spiral for every one that we had in the second, but this is not what we see.

Sometimes there are two circles next to each other at the same level of utility but with increasing loiter time and cost, sometimes three. This is typical behavior of tradespaces and is explained by the data given by running the model: at least with the assumptions that have gone into this model, it is fairly easy to keep the bomb in the air with a small jet engine. The maximum time aloft can easily exceed the amount that was initially deemed useful to the user; when this happens, the time aloft increases—increasing the size of the circles on the graph—but no extra utility is gained. The cost increases because the fuel adds weight to the architectures. This reflects the engineering that goes into "stretching" the bomb's frame to hold the extra fuel.

This interesting structure described in the previous paragraph might be able to tell us more. Here is the same graph with four "sister" architectures highlighted by plusses:




As can be seen here, there are four architectures highlighted. They represent four architectures in the third spiral that came from a single architecture in the first and second spiral. Architecture #1 has 10 pounds of fuel, #2 has 15, #3 has 20, and #4 has 25 pounds of fuel. 10 pounds of fuel is unable to propel this architecture to achieving full loiter utility, but 15 pounds is able to do so. Since full loiter utility is achieved with 15 pounds of fuel, 20 and 25 pounds are simply overkill.

The question could be asked whether or not the utility bounds were selected properly, i.e., would the user **actually** get more utility out of the bomb's being able to stay aloft longer? The

bounds were, in fact, defined incorrectly: when asked, the surrogate user explained that while the bomb is vulnerable to being shot down while aloft, a longer loiter time is always more useful. These confusions on limits are common and come from either lack of communication between the user and the systems architect or from the user's pre-formed belief that they are giving the value that they believe is the maximum possible even though they would actually have utility if something larger is possible. This latter is due to the lack of familiarity with the process; in the past, users have taken as much as two or three months to "get it." (Ross, '03) Once they have gotten it, though, they begin to see what the MATE system does is translate what they desire into technical jargon for the engineer, and translate what the system is designed to do into "user jargon" for ease of understanding. At its heart, MATE is a communication tool.

In this case, normally another interview would be preformed with changed boundaries on loiter time. Lack of time is the only reason that a second interview was not completed and utilized in building a new tradespace. This is an excellent example of how easily an engineer can mistakenly come to a conclusion: the upper bound of 40 minutes was determined through increasing LOCAAS's estimated maximum time aloft of 30 minutes. Verification of the tradespace by consulting the user helps the engineer build knowledge and intuition about the system and the user's desires.

While it is technically easy, according to these models, to accomplish the maximum utility in this case, adding this attribute changed our optimal solutions a great deal, as we shall see in the next section.

### 5.3.4 **Analysis by Specific Architectures**

It is expected to find four different architectures for each one of the five original architectures that were chosen. As has been seen from the analysis, it is easily possible that each of the sets of

four architectures may have the same utility. However, this is not the case. First, we present a chart of the architectures. Each previous architecture was labeled 1, 2, 3, 4, or 5; the corresponding four architecture variants are labeled here as 1a, 1b, etc.:

| | Arch # | Utility | Cost (Normalized) | Stand-off Dist. (nmi) | Stand-off Utility | Sort-ies | Sorties Utility | Re-target time | Re-target utility | Loiter time | Loiter utility |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1a | 1795 | 0.7857 | 1.0475 | 20.52 | 0.0111 | 3 | 0.9583 | 4 | 0.4500 | 24.06 | 0.5370 |
| b | 1802 | 0.8362 | 1.0496 | 20.52 | 0.0111 | 3 | 0.9583 | 4 | 0.4500 | 35.49 | 0.8710 |
| c | 1809 | 0.8557 | 1.0517 | 20.52 | 0.0111 | 3 | 0.9583 | 4 | 0.4500 | 46.54 | 1 |
| d | 1816 | 0.8557 | 1.0538 | 20.52 | 0.0111 | 3 | 0.9583 | 4 | 0.4500 | 57.25 | 1 |
| 2a | 8067 | 0.9154 | 1.1641 | 33.62 | 0.7588 | 3 | 0.9583 | 2.6 | 0.5550 | 34.62 | 0.8462 |
| b | 8074 | 0.9331 | 1.1697 | 33.62 | 0.7588 | 3 | 0.9583 | 2.6 | 0.5550 | 50.97 | 1 |
| c | 8081 | 0.9331 | 1.1754 | 33.62 | 0.7588 | 3 | 0.9583 | 2.6 | 0.5550 | 66.74 | 1 |
| d | 8088 | 0.9331 | 1.1810 | 33.62 | 0.7588 | 3 | 0.9583 | 2.6 | 0.5550 | 81.96 | 1 |
| 3a | 17923 | — | 1.3422 | 46.83 | 0.9476 | — | — | 4.0 | 0.4500 | 40.48 | 1 |
| b | 17930 | — | 1.3532 | 46.83 | 0.9476 | — | — | 4.0 | 0.4500 | 59.48 | 1 |
| c | 17937 | — | 1.3641 | 46.83 | 0.9476 | — | — | 4.0 | 0.4500 | 77.73 | 1 |
| d | 17944 | — | 1.3750 | 46.83 | 0.9476 | — | — | 4.0 | 0.4500 | 95.28 | 1 |
| 4a | 17926 | 0.9473 | 1.5494 | 46.83 | 0.9476 | 2 | 1 | 2.6 | 0.5550 | 33.94 | 0.8268 |
| b | 17933 | 0.9647 | 1.5663 | 46.83 | 0.9476 | 2 | 1 | 2.6 | 0.5550 | 49.79 | 1 |
| c | 17940 | 0.9407 | 1.5833 | 46.83 | 0.9476 | 2 | 1 | 2.6 | 0.5550 | 64.96 | 1 |
| d | 17947 | 0.9407 | 1.6002 | 46.83 | 0.9476 | 2 | 1 | 2.6 | 0.5550 | 79.49 | 1 |
| 5a | 32263 | 0.9178 | 2.0463 | 60.42 | 1 | 4 | 0.9167 | 2.6 | 0.5550 | 31.13 | 0.7377 |
| b | 32270 | 0.9464 | 2.0773 | 60.42 | 1 | 4 | 0.9167 | 2.6 | 0.5550 | 45.55 | 1 |
| c | 32277 | 0.9464 | 2.1083 | 60.42 | 1 | 4 | 0.9167 | 2.6 | 0.5550 | 59.29 | 1 |
| d | 32284 | 0.9464 | 2.1391 | 60.42 | 1 | 4 | 0.9167 | 2.6 | 0.5550 | 72.39 | 1 |

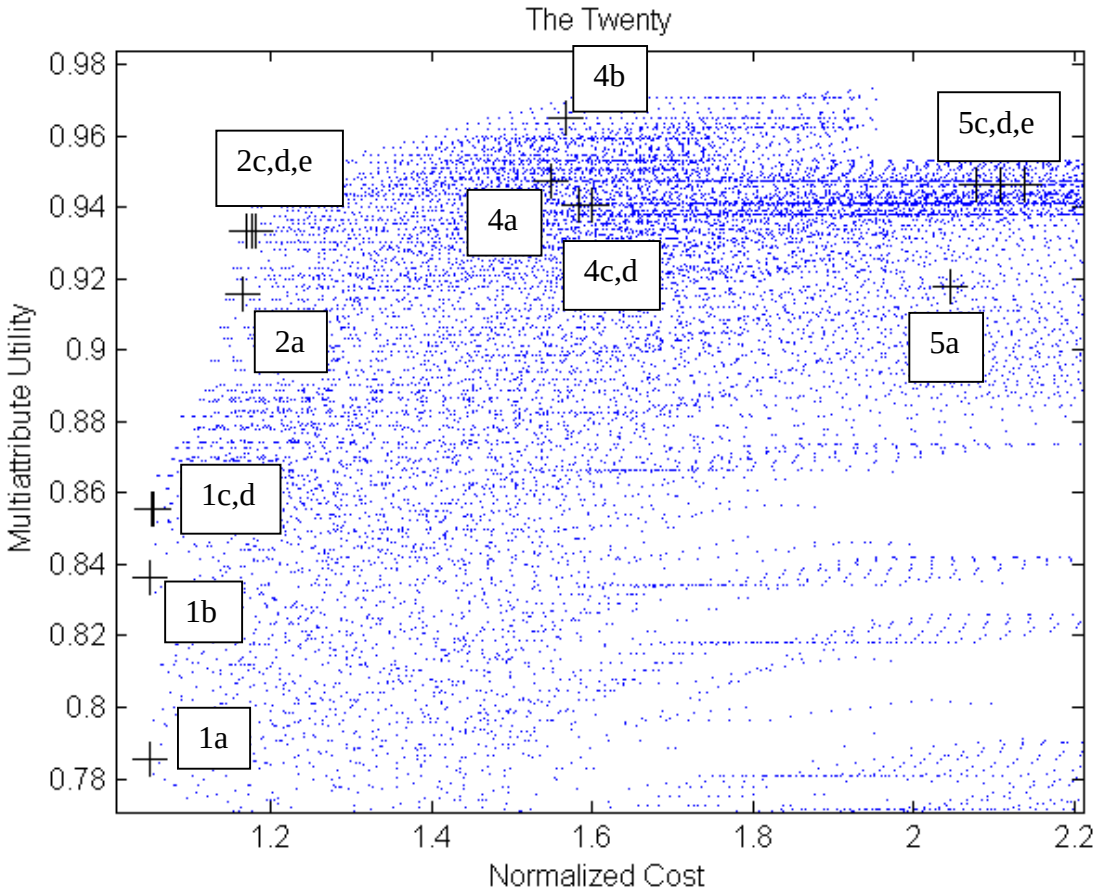
Two things immediately jump out of this chart: first is that many of the loiter utilities are almost all one. This is exactly what we suspected from the previous analysis of the tradespace: as the bomb achieves the maximum loiter time, the model says that the user will receive no extra utility from continued loitering.

Second, and far more interesting, is that some architectures fail to achieve the minimum allowable success level in the attribute of "Sorties to Destroy Target Set," and thus fail to

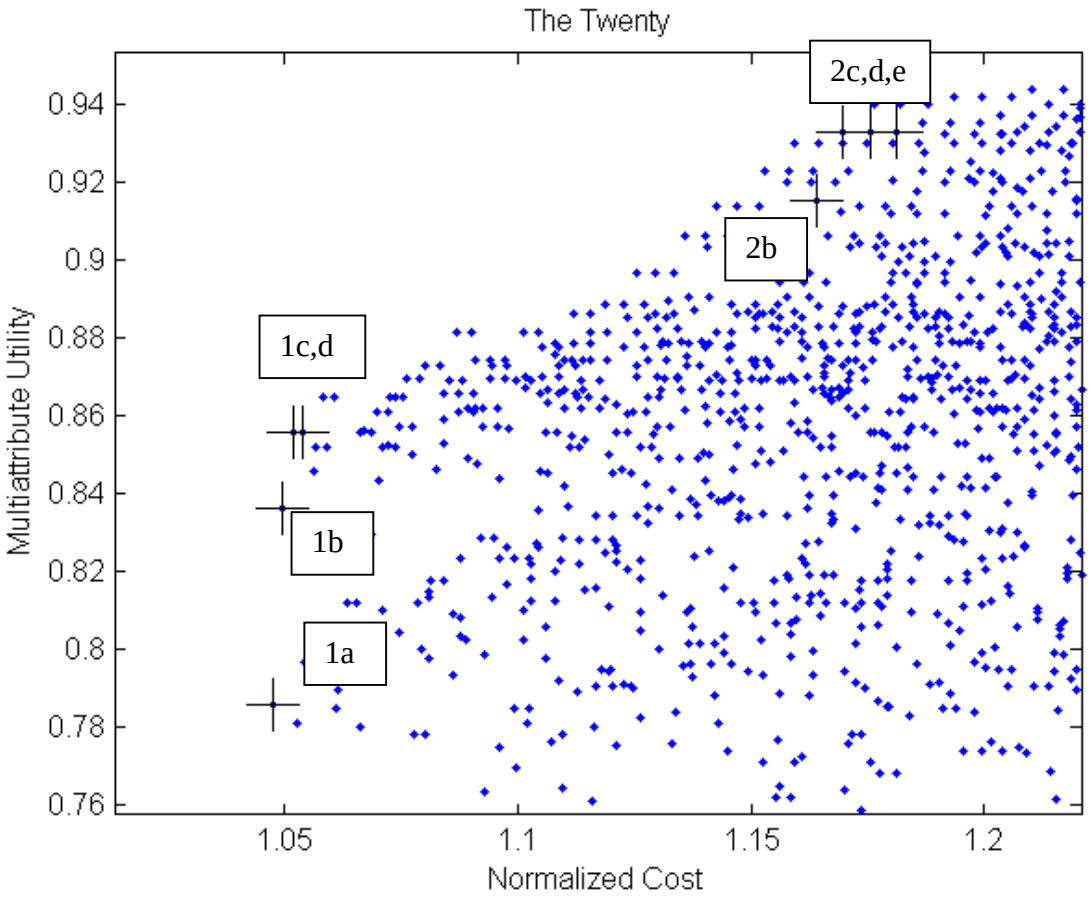
complete their prescribed mission. This is due to the increased size of the bomb from the fuel weight being added, which causes the bomb to be too long to fit into the F/A-22's bomb bays.

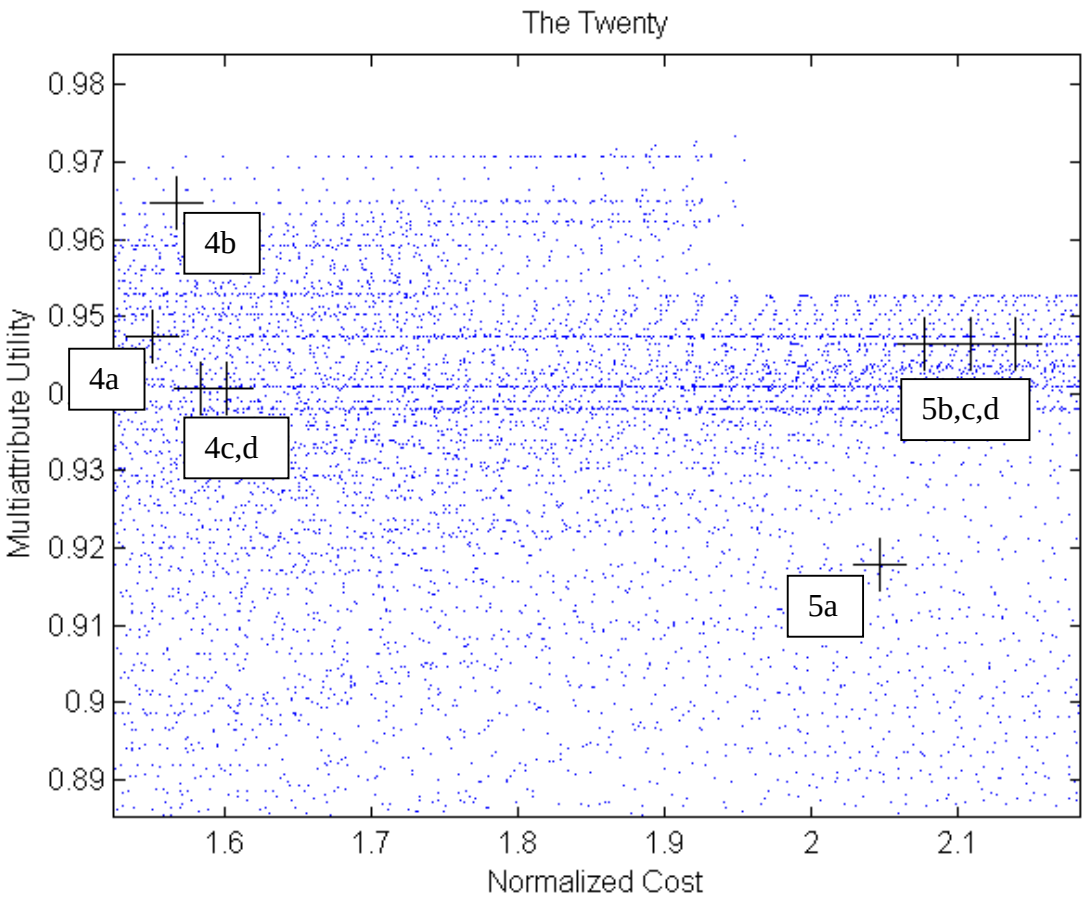
What is most interesting about this particular set of architectures is that the ones that fail to fit into the F/A-22's bomb bay are all descended from a single architecture—architecture #2,505—point #3 in our previous graphs. This is an important factor to consider when choosing an initial architecture for the first spiral. Since new or improved missions will definitely be added for new spirals, the systems engineer should know what parts of his system to make more flexible, what choices will impact possible future missions and how.

The plot below shows different interesting things:



There are many groups of architectures, but they are not evenly distributed. What is interesting about these architectures is that they are almost all sub-optimal. As shown by the enlargements below, only five architectures are still on the pareto-frontier; the other ten are not. In most of these cases, each of the architectures chosen because it was on the pareto-optimal frontier in the first spiral is outclassed by at least one other architecture, sometimes by several. In the case of the most expensive and highest utility architectures, they are outclassed by literally dozens or even hundreds of other architectures.






The underlying physical nature of these bombs has changed for the third spiral: a fuel tank has been added, "stretching" the bomb. The table of physical quantities for the architectures has added the varying fuel weights to it:

| | Old Arch. # | New Arch. # | Aspect Ratio | Guidance Type | Num. of Actuators | Explosive Weight | Casing Weight | Dia-meter | Fuel Weight |
|---|---|---|---|---|---|---|---|---|---|
| 1a | 451 | 1795 | 0.20 | Inertial | 2 | 150 | 12.5 | 5 | 10 |
| b | 451 | 1802 | 0.20 | Inertial | 2 | 150 | 12.5 | 5 | 15 |
| c | 451 | 1809 | 0.20 | Inertial | 2 | 150 | 12.5 | 5 | 20 |
| d | 451 | 1816 | 0.20 | Inertial | 2 | 150 | 12.5 | 5 | 25 |
| 2a | 2019 | 8067 | 0.55 | Inertial | 2 | 150 | 34.4 | 5 | 10 |
| b | 2019 | 8074 | 0.55 | Inertial | 2 | 150 | 34.4 | 5 | 15 |
| c | 2019 | 8081 | 0.55 | Inertial | 2 | 150 | 34.4 | 5 | 20 |
| d | 2019 | 8088 | 0.55 | Inertial | 2 | 150 | 34.4 | 5 | 25 |
| 3a | 4483 | 17923 | 1.10 | Inertial | 2 | 150 | 68.8 | 5 | 10 |
| b | 4483 | 17930 | 1.10 | Inertial | 2 | 150 | 68.8 | 5 | 15 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | 4483 | 17937 | 1.10 | Inertial | 2 | 150 | 68.8 | 5 | 20 |
| d | 4483 | 17944 | 1.10 | Inertial | 2 | 150 | 68.8 | 5 | 25 |
| 4a | 4486 | 17926 | 1.10 | Inertial | 2 | 150 | 110.0 | 8 | 10 |
| b | 4486 | 17933 | 1.10 | Inertial | 2 | 150 | 110.0 | 8 | 15 |
| c | 4486 | 17940 | 1.10 | Inertial | 2 | 150 | 110.0 | 8 | 20 |
| d | 4486 | 17947 | 1.10 | Inertial | 2 | 150 | 110.0 | 8 | 25 |
| 5a | 8071 | 32263 | 1.90 | Inertial | 2 | 150 | 213.8 | 9 | 10 |
| b | 8071 | 32270 | 1.90 | Inertial | 2 | 150 | 213.8 | 9 | 15 |
| c | 8071 | 32277 | 1.90 | Inertial | 2 | 150 | 213.8 | 9 | 20 |
| d | 8071 | 32284 | 1.90 | Inertial | 2 | 150 | 213.8 | 9 | 25 |

Each architecture's "heritage" can be seen clearly in this table: every set of four architectures is exactly the same except for the fuel weight vector. This vector expands each previous architecture into four, one for each unique fuel value. The heritage architecture is labeled as "Old Arch. #," and the architecture numbers that result are in labeled as "New Arch. #," so the reader can cross-index between this table and the utility table.

It is again important to remember that, while the underlying physical quantities are important for the engineer, it is the utility numbers that will help him communicate to the user. In fact, the utility numbers—both the multi-attribute utility numbers and the single-attribute utility numbers—are calculated from and represent the underlying physical architecture. As they put the physical architecture into the user's jargon and terminology, they are what this paper has concentrated on.

# 6 Discussion & Implications

## 6.1 The Difficulties of Evolutionary Acquisition

There are many challenges that a program manager must face when he or she works on an Evolutionary Acquisition system. Not the least of these challenges are the decisions made as to what the next step in the Evolution will be.

Unfortunately, the decision analysis methods used are often poor. Add to these poor analysis methods the immense complexities of spiral development, and the result has potential for sub-optimal design or even mission failure.

Worse yet is the method of conveying what the system is supposed to do: requirements. Requirements are a translation into technical language of what the user actually wants. Any time that there is a translation from one dialect to another, even though the people involved speak the same language, there is potential for error and frustration.

Many of these problems are getting better. There are many intelligent people in the acquisition sector and in aerospace in general. However, the problems arise because of company procedures and previous problems; it often takes disruption—or the help of an external body such as the Lean Aerospace Institute of MIT—to provide the impetus, or ability and knowledge, to change for the better.

## 6.2 Multi-Attribute Tradespace Exploration

MATE gives us a measure of overall success of the mission, **as defined by the user or decision maker.** While the process is initially difficult for the user, as shown by the surrogate user, once he or she is familiar with it, new interviews can be done in a matter of a few dozen minutes.

Meanwhile, the system engineer is using her utility function to guide the design to perform at the best level possible for the amount of money the decision maker is willing to spend.

In addition, the user is given a suite of possible optimal architectures and allowed to choose what he or she believes is best for the system and the group of higher systems that it will fit into. In the Air Force, this means that he or she can built warplanes and weapons that fit well into the existing framework; in general, it is possible for the user to consider how various formats of the system under consideration will fit into the overall company's mission.

### 6.2.1 **Building Intuition**

The utility function helps the engineers design not just an optimal architecture, but it actually helps them build insight into what the system is meant to do. In fact, the users that have participated in this process have often found that the process helps them understand both what they want, but how it is provided for them. By fostering communication between engineers and scientists, between the builders and the buyers, between the government and the contractors, MATE can potentially help the process along.

MATE fosters communication through its translating ability. For the user, the results are put into terms that he or she defined; for the engineer, the technical parameters are fed through the model to arrive at these results. Thus, each side is "spoken to" in its own language. This not only helps the communication to occur, but enforces a common vocabulary throughout. As the user and the engineer become more familiar with what the former wants and the latter can provide, several optimal architectures can be examined thoroughly and a down select made to a single point design architecture that is best for the current circumstances.

### 6.2.2 **Showing What is Possible and Problematic**

Of course, by helping to quantify the decisions being made, MATE also helps the decision maker make the best possible decision. Rather than choosing what he or she wants at the beginning without any knowledge of what is possible, the MATE process helps the decision maker by showing him or her what is possible before the decisions are made.

Imagine, for instance, if a decision maker knew that he or she could use a smaller engine on the third spiral of the SDB system and still receive a loiter time of 40 minutes before any money had been committed, before any materials or manufacturing processes had been determined? Or that adding an engine to an already expensive and high-utility architecture might keep it from fitting into the F/A-22, again before the bulk of the system cost had been determined?

Of course, the four failed architectures in the third spiral that were not successful were only a small number of the failures. Of the 34,944 architectures, more than one third of them fail for one reason or another. Why this happens is complex, but can be determined. Either the bomb is too big to fit into the bomb bay, or the wings are too stubby to achieve the minimum glide distance, or something else. The systems engineer can determine these causes of failure and through this knowledge help the user understand what is and is not possible.

### 6.2.3 **Highly Applicable**

Finally, MATE is highly applicable. It has successfully been used on satellites and bombs; it is based on the MAUA method developed by Keeney & Raiffa, a method that has been used successfully on everything from airport placement to fire engine route optimization. MATE adds in the enumeration of the tradespace based on mathematical modeling; anything that can be modeled can be represented in the MATE system.

### 6.2.4 **Specifically Applicable to Evolutionary Acquisition**

MATE was applied to three spirals of the Small Diameter bomb system by a single graduate student over the period of approximately nine months to a year who had no previous experience with weapons systems and was taking classes the entire time. This author's knowledge of bomb systems, and especially of the planned system of the Small Diameter Bomb, has been enhanced significantly. Adding up the three spirals, a total of 52,416 different architectures were created; five particular choices were carefully examined and tracked through three spirals. The results were interesting and potentially useful for the community engaged in the creation of the Small Diameter Bomb. These results may have been particularly useful before the extensive Analysis Of Alternatives preformed by the Air Force over two years.

The implications of this process are staggering. If this can be done by a single person with no experience who is under external time constraints, imagine what might be possible with experienced engineers working full time on this process? More accurate models could be built; wind tunnel tests on physical models of the system could provide accurate aerodynamic data; in a relatively short period of time, a group of engineers could provide a decision maker with data that would greatly help him or her to decide how the spirals would proceed. If the world situation or the funding situation changed, the models are easily modified and reused to again assist the decision maker. In this way can a balance be achieved between planning ahead and taking account of constraints applied by the current environment. Incorporating experience from the use of existing architectures can be incorporated easily into the MATE models to again assist the decision maker with his or her decision of what to do with the system.

Again, the benefit to using MATE is not only that the user finds out what the most optimal architectures are, but rather that it is a speedy process that is capable of change that give

a single overall metric to represent user utility. By focusing the engineers onto what the user actually wants rather than creating minimum requirements, and through increased communication between the design team and the end user, MATE can help a better product to be designed and built.

In short, it is this author's belief that not only is the MATE procedure useful for Evolutionary Acquisition, it's use—or the use of a system like it—may be a necessary (but not sufficient) component for success.

## 6.3 Limitations

### 6.3.1 **Limited in number of attributes**

The rule of seven limits us to seven attributes, plus or minus two. This limitation is due to the determination of the k-values. To rank the k-values and determine their values, the interviewee must hold all of the different values in his or her mind and determine whether one configuration of those values is more desirable than another configuration of those values. Unfortunately, the fact is that the human mind is not able to consider more than seven items simultaneously.

Normally, there are ways around this for complex systems, but the MATE interview process precludes many of them. It is easy to imagine a system that is complex having seven attributes; in fact, many of the previous, space based, uses of MATE had seven attributes. If this system is to be applied to spiral development, it is difficult to imagine that it would be applicable to several spirals.

There turns out to be a way around it, however. What can be done is to nest utility functions. Imagine, for instance, that a first spiral design had seven attributes. The systems engineer could create a model to represent this system, but would have to get creative if the next

spiral added more attributes. For the sake of argument, let us say that the user desires another four missions for the second spiral. What can be done, then is to create a five attribute MATE model with four of the attributes being new attributes that the user desires for the new spiral, and one attribute that is based on the multiattribute utility from the first spiral.

This "nested" utility function would have to be interviewed for by the user. He or she would have to determine how much use the old utility numbers were in the new system. However, having done the MATE process with the old system, he or she will have a great deal of intuition about the system, and might be able to accomplish this interview. Since it has never been tried, of course there is no guarantee of success. The idea seems quite complicated and may in fact be impossible.

### 6.3.2 **Computing**

The SDB is a relatively simple system, and yet it nearly got out of hand during the third spiral. Running the model on the computers available to this author, it took as much as five minutes to create and save a graph, or about four to five minutes to run the program to determine the tradespace.

If there had been an addition to the second spiral's Design Vector that had four or five possibilities, the total number of architectures would have grown to well over 135,000; the computers available in this author's laboratory would have run out of memory. Data analysis cannot be done if there are no computers able to hold all of the data.

There are methods of programming around such problems; not all of the modules built for this model were functions, and functions save computer memory. Another strategy would be that each part of the final tradespace could be saved to disk individually, the memory wiped, and the tradespace concatenated together after all the calculations had been done.

However, while these methods get around the problem initially, they cannot overcome the complexity of a larger system. If, for instance, there were a few more spirals, the tradespace of the SDB would quickly grow to $10^6$ or even $10^7$ architectures. Computer size and time can be a severely limiting factor to representing a system accurately.

This can also happen in a more complex system for a single development cycle. For instance, imagine a complex system that has seven attributes, each with forty possibilities. This results in 163,840,000,000 architectures, each with its own detailed description in computer memory. Based on the far more simple SDB model, in which each architecture takes up approximately 600 bytes of memory, a computer would need to have at least a terabyte of RAM just to hold the tradespace, not to mention two or three terabytes to do the calculations efficiently.

### 6.3.3 **Modeling**

Mate can only be used for systems that the engineers on the project know how to model. Truly wild solutions may not be easily modelable. Engineering teams are under time constraints, and will likely not pursue ideas that they do not know how to model. This could lead to sub-optimal designs and lack of creativity in problem solving. Unfortunately, the only way around this is for the systems engineer or the team manager to be aware that it may be a problem and guard against it. Education and experience are invaluable for knowing what is possible and what is not.

Even if models can be written, another problematic part of modeling is the accuracy of new models. In this case, there were applicable models available that were well known, such as the Breguet range equation. Space systems are highly modelable, and so MATE is particularly applicable to them. Anything that does not have standardized models may run into accuracy problems as new models are invented.

There are even areas where MATE may not be applicable at all: areas such as computer design, where the pace of technological change is so high that even if a model is made, it is obsolete by the time the engineer has verified that it is relatively accurate. Another area would be wild designs on frontier technologies. Models don't exist for technologies that haven't been used before, and using these new technologies in ways that they were not designed for are likely to be completely unmodelable.

In cases such as these, MATE is not likely to be useful. In the latter case of wild designs with frontier technology, prototyping may be the only way to determine if the system has merit. With rapidly changing technology, a modular architecture with a well known computer language may allow newer, better components to be swapped in for old.

These are concerns that are probably not apropos for Evolutionary Acquisition, though, since EA relies on mature technologies. These technologies should be modelable by the time they are used in EA systems.

### 6.3.4 **Time, Scheduling, and Frustration**

Time plays another important role in limiting the MATE system. It may be difficult to get the time needed with the user or decision maker to explain what MATE is and does; it may be more difficult to get the time that it takes for that person to become familiar enough with MATE to see why they should spend more time than usual.

The interview process takes time for the interviewee to become familiar with it. When these interviews are done face to face, they can be quite intimidating. The interviewee must be prepared for questions that seem unanswerable at first. MATE challenges scientists and other users to use a mode of thinking that they have never used before. While it may sound easy on

paper, it is in fact quite difficult to ignore what you think to be the upper limits of success and concentrate only on what you desire.

Interestingly, the process is much easier with a computerized interview system. It is believed among the people who have designed the MATE process (primarily Ross and Diller) that this is due to the absence of the pressure that having another person present causes. There is no need to hurry if nobody is waiting for you; there is no need for embarrassment if nobody sees that it takes you a half an hour to understand how a question is being put to you.

As stated in the previous section, however, once the interviewee becomes familiar with these ideas, he or she can become an interview pro. Near the end of this process, the surrogate user was able to complete interviews within the space of an hour.

### 6.3.5 **k-value Confusion**

While the k-values are often taken as weighting values, it is not entirely true that an attribute with a higher k-value is necessarily more important than another. As explained in Keeney & Raiffa, these k-values work with the system model to determine the utility for that particular value of that attribute. The model, what is being modeled, the upper and lower boundaries placed on what is being modeled, and the k values all contribute to the multi-attribute utility numbers. This can cause confusion among users of the system, especially when only glancing at a tradespace instead of thoroughly examining it.

This is exemplified by the examination of the SDB tradespace when the circles are sized by retarget time. This particular instantiation of the tradespace and the analysis preformed on it show that retarget time is a tertiary driver even though it is the main purpose for building the second spiral. It seems as if something that is unimportant is being added, when, in fact, what is happening is that the smaller k-value placed on that attribute is combining with an extremely

small difference in the actual attribute combining with the nearly flat utility curve combining with the predicted relative ease of accomplishing the physical mission of retargeting the bomb. When these things combine, they downplay the role of retarget time even though its k-value is only 0.1 smaller than standoff distance's k-value.

Since standoff distance remains one of the primary drivers of utility throughout the three spirals, so it would seem that retarget time should also be a primary driver, or at least a secondary driver. However, standoff distance's attribute values vary more than retarget time's, the user's utility curve is not nearly as flat, and the mission is more difficult to achieve. These factors combine with the slightly higher k-value to make standoff distance much more important in the model.

Still, the model seems to be valid. These things must be examined and questioned, and preferably shown to the user for verification. This educates the user in the tightly coupled interactions of the system while verifying the engineer's models—in short, it increases communication.

## 6.4 Further Research Opportunities

This research has raised more questions in the author's mind than it answered. Some of these questions are worth of further research.

### 6.4.1 **Modeling and Optimizing Different Possible Spiral Paths**

One thing that is very intriguing is the idea that different paths can be taken through the spirals. As a hypothetical example, it might be more expedient to build the engine in for loiter time in the second spiral and wait for the technology to develop for retargeting systems.

Retargeting a bomb is easy in the physics of the thing, but quite difficult in the programming. Automatic retargeting is the ideal way to employ such a technology; otherwise, there must be an operator giving new coordinates to the bomb. It is hard to imagine that this would be feasible for some of the architectures considered in this paper—in architecture #34,940, for instance, 318 bombs fit into the bomb bay of the F/A-22. If all of those bombs were dropped at once, the ability to retarget them would require 318 operators tracking each bomb's progress towards its initial target, which seems unlikely.

Thus, automated retargeting would be ideal. This would require automatic target recognition, which is quite difficult in practice. Computer "vision" is not able to tell the difference between a jeep filled with civilians and a jeep filled with soldiers; nor is it able to tell the difference between a building and a bunker. Perhaps a movement sensor could be put in, so that the bomb would retarget only to moving targets; perhaps it would only retarget if it recognized a target that can only be a military target, such as a tank.

This technology is expected to improve over time. By pushing it off until the third spiral, it might cost less or be more effective. It is possible to weight the architectures' cost numbers—or, with a little more math, their utility numbers—and discount the utility gained. Using such discounted numbers, various different models of paths through the spirals could be run to determine which would bring the most utility to the user, most quickly, and with the least amount of resources spent.

### 6.4.2 **Modeling and Optimizing Different Possible Spirals**

What if there were more uses for the system? What if there were multiple users, and each one wanted a specific capability to be added to the system? Could a Program Manager use MATE to

decide which of these capabilities was most useful for both the immediate and long term future of the system?

### 6.4.3 **Combining Search Techniques (Cyrus Jilla, MIT thesis 2002)**

Cyrus Jilla's work in partial tradespace enumeration is certainly applicable to larger MATE systems. In his thesis (A Multiobjective, Multidisciplinary Design Optimization Methodology for the Conceptual Design of Distributed Satellite Systems, MIT 2002) Jilla demonstrates several different techniques for intelligently enumerating only a small part of a tradespace but still finding the optimal solution. Jilla's work might be extendable to finding the pareto-optimal frontier, which is key for the proper decision assisting powers of MATE. If this were possible, it would be beneficial to MATE because it would significantly reduce calculation time, thus allowing more complex modeling and enumeration of larger tradespaces.

In addition to significantly reducing calculation time, it would also significantly reduce the amount of memory needed to find the optimal frontier. Instead of needing a terabyte of memory for a complex system, it would be possible to run the same model with only a gigabyte of memory. While this is a lot of memory, it is something within the realm of possibility within the next few years. Supercomputers may already be at this point; to the author's great disappointment, he does not know from personal experience whether or not such computers are extant now.

### 6.4.4 **Using Actual Decision Makers and Improving the Models**

It was unfortunately not feasible to use real decision makers for this exercise. If one could be found to re-take the interviews, it would be an interesting study to see how this would affect the tradespaces.

Another possibility would be to improve the fidelity of the models. Adding in the actual shape of the F/A-22's bomb bays and writing a bomb placement optimization routine would increase the actual usefulness of this code to the Air Force. Wind tunnel testing on various shapes to determine their actual L/D would improve the fidelity of the aerodynamic estimations made in this paper.

### 6.4.5 **Combining Multiple Stakeholders**

This system could benefit greatly from analysis of data from multiple stakeholders. There are three stakeholders that would be useful: first, the pilot user would be interviewed. Second, the warfighter/general user would be interviewed.. Finally, the decision maker would be interviewed.

It is unclear how these sets of data would be incorporated. There are several ideas in Diller '02. The most promising of these is nested utility functions. In this sort of analysis, the decision maker has a set of attributes that include system attributes such as cost and schedule time, but there are also attributes where each of the USER's utilities are measured. Thus, the user's overall utility—composed of several attributes—becomes a single attribute within the decision maker's utility function. Thus, an overall utility function can include utility functions of subsystem engineers, users, or any other utility functions.

One possible political problem with this method is that, if there were multiple users, the k-values could be seen as the decision maker's opinion of who is more important than whom. While this could be a political problem, it is a real-life common practice to value one user over another. This is done for various reasons, but it is nevertheless so.

Another possible method is, for two user utilities, to graph the tradespaces on three dimensional graphs. This does not seem profitable, as three dimensional graphs are very difficult to analyze well.

Another possibility would be to give the interviews to a group of people. It has been proven that group preferences are transitive (Ross, '03), but it might be possible to get a group together once to determine the utility curves and never actually update those curves because it is well known that those updates will change because the group's preferences have changed.

## 6.5 Research Questions

Near the beginning of this thesis, there were four overarching questions that drove us toward this exploration of MATE in the use of Evolutionary Acquisition. They were:

1) Can Multi-Attribute Tradespace Exploration be used in a non-space application?
2) Can MATE be useful for Evolutionary Acquisition?
3) Does MATE help in gaining insight into the system being modeled?
4) Can MATE be used to track architectures through different Evolutions?

Although not rigorously tested, it seems clear that the answers to these questions are positive. MATE has been successfully applied to the Small Diameter Bomb, a non-space system. Problems were found, but most of the problems were modeling problems by nature. Models were found or made, and the analysis of the system went on.

Evolutionary Acquisition presents its own challenges: speed of design, quick change, high levels of communication, and concentration on system-level concepts; MATE is a tool that can potentially help the systems engineer with each of these in turn: although time pressures still impinge upon perfect use of the MATE tool, such as with the loiter time attribute in this paper, the speed of model building and quick changes to those models help create designs quickly. The necessity of a high level of communication to begin and verify the models forces the engineer

and the user to use the same jargon, greatly facilitating further communication between them. The abstraction of the architectures into utility and attributes creates a sense of how the architecture performs in these same broad areas, thus facilitating a systems-level mode of thinking.

MATE's concentration on the customers preferences and the way that those preferences rule the design of the system help the engineers gain insight on the system and the user's preferences. Finally, we have seen how tracking different architectures through different spirals can help the systems engineer to understand why architectures should or should not be chosen, where flexibility is important, and what the system and utility drivers are.

### 6.5.1 **Conclusion**

Attributes of a system that allow flexibility are good; knowing in what way to make a system flexible allows that flexibility to be applied consciously and utilized more effectively than simply adding excess capacity. Modeling the Small Diameter Bomb has taught this author much about not only the SDB, but also about MATE itself. MATE has its limitations and difficulties, just like any other tool in the engineer's toolbox, but its benefits are excellent despite these problems.

The main thing that I have learned about MATE is that it is a tool for communication between user or decision maker and engineer. By fostering communication and understanding, it is my belief that superior products can be made with the help of the MATE system.

For evolutionary acquisition, using the MATE process to plan spirals can help engineers and decision makers to more fully understand what will be necessary for the initial system to enable its growth throughout its lifetime. MATE is an excellent tool for the systems engineer faced with the monumental challenges of evolutionary acquisition.

# A. Appendix: Matlab Code

## 1. First spiral

### a) SDBMain.m

```
%--------------------------------------------------------------------------------
% SDB Main
% This file calls all the other files in the correct order
% Author: Jason Derleth
% ate: 8/13/2002
% Version: 0.1
% Inputs:
%   none
% Outputs:
%   none
%--------------------------------------------------------------------------------

clear

SDBConstants                    %Calls constants into workspace
SDBDesignVector                 %Calls the DesignVector into workspace
SDBGlideDistance                %Calculates glide distance and L/D
SDBCrossSectionalDensity        %Calculates X-sectional density and length of bomb
SDBCircularErrorProbable        %Calculates Circular Error Probable
SDBNumberInFA22                 %Calculates how many SDBs will fit into an FB-22
[Tradespace] = SDBSortie(Tradespace,Constants,DesignVector); %Calculates number of sorties
necessary to destroy target set
SDBCostSchedule                 %Estimates Cost and Schedule of bomb--should be made
relative cost and schedule
[Tradespace] = SDBUtility(Tradespace,Constants);      %Calculates Utility
```

### b) SDBConstants

```
%--------------------------------------------------------------------------------
% SDB Constants
% This file initializes needed constants
% Author: Jason Derleth
% Date: 8/21/2002
% Version: 0.1
% Inputs:
%   none
% Outputs:
%   Constants struct
%--------------------------------------------------------------------------------

Constants.AverageBombDensity = 0.06;       %0.174 lbs/in^3 is the highest existing bomb
density
```

```
Constants.LaunchSpeed = 750;                %In MPH
Constants.LaunchAltitude = 20000;           %In feet
Constants.DragCoefficientSub0 = 0.0045;     %Estimation from p.278 of Raymer 1st ed.
Constants.ProductionNumber=5000;            %Estimate on number to be produced in 5 years
Constants.TestBombs=10;                     %Estimate on number of flight tests to be used
Constants.LayersInBay=2;                    %in layers of bombs
Constants.BombBayWidth=30;                  %in inches
Constants.BombBayLength=18*12;              %in inches
Constants.NumBomBays=2;
Constants.NumAttributes=3;
%Constants.Smallk=[.275 .375 .475 .300];         %input from Excel sheet
Constants.Smallk=[.3 .700];         %input from Excel sheet
Constants.BigK=nirav_calculate_K(Constants.Smallk);   %Calculates Big K
Constants.BunkerKillDistance=1;             %in meters
Constants.TankKillDistance=3;               %in meters
Constants.APVKillDistance=4;                %in meters
Constants.JeepKillDistance=5;               %in meters
Constants.NumberOfBunkers=5;
Constants.NumberOfTanks=2;
Constants.NumberOfAPVs=5;
Constants.NumberOfJeeps=10;
Constants.AssumedKillPercent=0.95;          %Percentage where kill is assumed
```

**c) SDBDesignVector**

```
%-------------------------------------------------------------------------------
% DesignVector Enumeration
% This file enumerates the DesignVector for the Small Diameter Bomb Code
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.1
% Inputs:
%   None
% Outputs:
%   DesignVector struct:
%       DesignVector(i).AspectRatio          Ratio used to calculate lift, drag, glide distance...
%       DesignVector(i).Guidance             Either Inertial, GPS, or Active
%       DesignVector(i).ExplosiveWeight      In pounds of explosive
%       DesignVector(i).CasingWeight         In % of Explosive Weight
%       DesignVector(i).Diameter             In inches
%-------------------------------------------------------------------------------

clear DesignVector
i=1;
for AspectRatio=[0.1:0.05:2]
   for Guidance={'inertial' 'GPS'}
      for Actuators=[2:3]
```

```
        for ExplosiveWeight=[150:10:220]
           for Diameter=[3:1:9]
              DesignVector(i).AspectRatio=AspectRatio;
              DesignVector(i).Guidance=Guidance;
              DesignVector(i).Actuators=Actuators;
              DesignVector(i).ExplosiveWeight=ExplosiveWeight;
              DesignVector(i).CasingWeight=ExplosiveWeight*…
                          (AspectRatio*10)*(Diameter/12)*.1;
              DesignVector(i).Diameter=Diameter;
              i=i+1;
           end
        end
     end
  end
end
```

**d) SDBGlideDistance**

```
%--------------------------------------------------------------------------------
% SDB Glide Distance Calculation
% This file calculates Glide Distance
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   DesignVector struct
% Outputs:
%   Tradespace.GlideDistance vector added on to Tradespace struct, all distances in mi
%   Tradespace.LiftOverDrag vector added to Tradespace struct
%--------------------------------------------------------------------------------


for i=[1:size(DesignVector,2)];

   % First, we must calculate e, the Oswald Span Efficiency:
   e=1.78 * (1-0.045 * (DesignVector(i).AspectRatio ^ 0.68)) - 0.64; % From p.297 of Raymer
1st ed.

   %Now, we calculate the Glide Ratio, i.e., how much do we drop in z for every distance in x:
   GlideRatio=0.5 * sqrt((pi * DesignVector(i).AspectRatio * e) /
Constants.DragCoefficientSub0); % From p.467 of Raymer 1st ed.

   %Given that and an initial height, we know how far the bomb will glide:
   Tradespace(i).GlideDistance = [(GlideRatio * Constants.LaunchAltitude) / 6076.115];  % in
nautical miles
   Tradespace(i).LiftOverDrag = GlideRatio;
End
```

**e) SDBCrossSectionalDensity**

```
%--------------------------------------------------------------------------------
% SDB Cross-Sectional Density Calculation
% This file calculates the Cross-Sectional Density for the Small Diameter Bomb Code
%          and also determines the length of the bomb.
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.0
% Inputs:
%   DesignVector struct
%   Constants struct
% Outputs:
%   Tradespace.CrossSectionalDensity
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
   Tradespace(i).CrossSectionalDensity=[(DesignVector(i).CasingWeight +
DesignVector(i).ExplosiveWeight) / DesignVector(i).Diameter];
       % CrossSectionalDensity in lbs/diameter
   Tradespace(i).Length=[(DesignVector(i).CasingWeight + DesignVector(i).ExplosiveWeight) /
(Constants.AverageBombDensity * pi * ((DesignVector(i).Diameter / 2)^2))];
       % Length in inches
   Tradespace(i).Diameter=DesignVector(i).Diameter; %in inches
   Tradespace(i).Wingspan=(Tradespace(i).Length)/24; %in feet, wings assumed to be 1/4 the
length of the bomb each
End
```

**f) SDBCircularErrorProbable**

```
%--------------------------------------------------------------------------------
% SDB Circular Error Probable
% This file assigns CEP based on the type of targeting system
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   DesignVector struct
% Outputs:
%   Adds Circular Error Probable, a vector, to the Tradespace struct
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      Tradespace(i).CircularErrorProbable = 13-2*DesignVector(i).Actuators;
   else
```

```
      Tradespace(i).CircularErrorProbable = 8-2*DesignVector(i).Actuators;
   end
end
```

**g) SDBNumberInFA22**

```
%-------------------------------------------------------------------------------
% SDBNumberInFA22.m
% This file calculates the number of SDBs that will fit into an FA22
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.0
% Inputs:
%   DesignVector struct
%   Constants struct
% Outputs:
%   Tradespace.NumberInFB22 vector added to Tradespace Struct
%-------------------------------------------------------------------------------

Margin=0.1; %ten percent margin on width and depth to account for mounting system

for i=[1:size(DesignVector,2)];
   HeightNum=Constants.LayersInBay; % LayersInBay is in number of layers of bombs that fit
in bay
   WidthNum=round((Constants.BombBayWidth*(1-Margin)/(DesignVector(i).Diameter))-0.5);
   LengthNum=round((Constants.BombBayLength*(1-Margin)/(Tradespace(i).Length))-0.5);

Tradespace(i).NumberInFB22=HeightNum*WidthNum*LengthNum*Constants.NumBomBays;
end
```

**h) SDBSortie**

```
function [outspace] = SDBSortie(Tradespace, Constants, DesignVector)
%-------------------------------------------------------------------------------
% SDB Sortie counter
% This file calculates the number of sorties needed to destroy the chosen target set.
% Author: Jason Derleth
% Date: 3/18/03
% Version: 1.0
% Inputs:
%   DesignVector struct
%   Tradespace Struct
% Outputs:
%   Tradespace.Sorties
%-------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
```

```
   if Tradespace(i).NumberInFB22==0
      Tradespace(i).Sorties=0;
   else
      ScalingFactor=sqrt(DesignVector(i).ExplosiveWeight/220);    %Divided by max explosive
weight
      BunkerKill=Constants.BunkerKillDistance*ScalingFactor;
      TankKill=Constants.TankKillDistance*ScalingFactor;
      APVKill=Constants.APVKillDistance*ScalingFactor;
      JeepKill=Constants.JeepKillDistance*ScalingFactor;
      sigma=Tradespace(i).CircularErrorProbable/0.68;          %Gives us sigma for normal
distribution
                                          %i.e., 50% of bombs fall within given CEP

      x=[-BunkerKill BunkerKill];
      P=normcdf([-BunkerKill BunkerKill],0,sigma);
      NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
         log(1-(1-(P(2)-P(1)))^2));
      NumPairs=round(NumPairs+0.5);
      NumBombs=2*NumPairs*Constants.NumberOfBunkers;

      x=[-TankKill TankKill];
      P=normcdf([-TankKill TankKill],0,sigma);
      NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
         log(1-(1-(P(2)-P(1)))^2));                        %guarantee assumed kill percentage
      NumPairs=round(NumPairs+0.5);
      NumBombs=NumBombs+2*NumPairs*Constants.NumberOfTanks;

      x=[-APVKill APVKill];
      P=normcdf([-APVKill APVKill],0,sigma);
      NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
         log(1-(1-(P(2)-P(1)))^2));                        %guarantee assumed kill percentage
      NumPairs=round(NumPairs+0.5);
      NumBombs=NumBombs+2*NumPairs*Constants.NumberOfAPVs;

      x=[-JeepKill JeepKill];
      P=normcdf([-JeepKill JeepKill],0,sigma);
      NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
         log(1-(1-(P(2)-P(1)))^2));                        %guarantee assumed kill percentage
      NumPairs=round(NumPairs+0.5);
      NumBombs=NumBombs+2*NumPairs*Constants.NumberOfJeeps;
```

```
      Sorties=(NumBombs/Tradespace(i).NumberInFB22);
      Tradespace(i).Sorties=round(Sorties+0.5);
      Tradespace(i).Fred=NumBombs;

   end
end
[outspace] = Tradespace;
```

**i) SDBCostSchedule**

```
%-------------------------------------------------------------------------------
% SDB Cost and Schedule
% This file calculates the cost and schedule based on the DAPCA IV CERs
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   Constants struct
%   Tradespace struct
% Outputs:
%   Adds TotalCost and TotalHours, vectors, to the Tradespace struct
%-------------------------------------------------------------------------------

EngRate=86;
ToolRate=88;
QCRate=81;
MfgRate=73;

 for i=[1:size(DesignVector,2)];
   TotalWeight=DesignVector(i).CasingWeight + DesignVector(i).ExplosiveWeight;

EngHours=(7.07*(TotalWeight)^0.777)*(Constants.LaunchSpeed^0.894)*(Constants.Production
Number^0.163);

ToolHours=(8.71*(TotalWeight)^0.777)*(Constants.LaunchSpeed^0.696)*(Constants.Productio
nNumber^0.263);

MfgHours=(10.72*(TotalWeight)^.82)*(Constants.LaunchSpeed^0.484)*(Constants.Production
Number^0.641);
   QCHours=0.133*MfgHours;
   DevelopSupportCost=(7.07*(TotalWeight)^0.630)*(Constants.LaunchSpeed^1.3);

TestCost=(1807.1*(TotalWeight)^0.325)*(Constants.LaunchSpeed^0.822)*(Constants.TestBom
bs^1.21);

MfgMaterialsCost=(16*(TotalWeight)^0.921)*(Constants.LaunchSpeed^0.621)*(Constants.Prod
uctionNumber^0.799);
```

```
   Tradespace(i).TotalHours=EngHours + ToolHours + MfgHours + QCHours;
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      GuidanceCost=4000*DesignVector(i).Actuators;
   else
      GuidanceCost=3000+4000*DesignVector(i).Actuators;
   end
   BombCost=.0000003423929*(ToolHours * ToolRate + MfgHours * MfgRate + QCHours *
QCRate + DevelopSupportCost + TestCost + MfgMaterialsCost);
   Tradespace(i).TotalCost=BombCost+GuidanceCost;
end
```

**j) SDBUtility**

```
function [outspace] = SDBUtility(inspace,Constants)
%-----------------------FUNCTION INPUTS-----------------------------------------------------
% Constants.BigK;                          %multiplicative constant          unitless
% Constants.Smallk;                        %single attribute constants       unitless
% CEP = inspace(i).CircularErrorProabable;   %Circular Error Probable         meters
% Standoff = inspace(i).GlideDistance;       %Standoff Distance              nmi
% Num = inspace(i).NumberInFB22;             %# That fit in FB-22             bombs
%-------------------------------------------------------------------------------------------


% %----------------------FUNCTION OUTPUTS-----------------------------------------------------
%  inspace(i).CEPUtility;                  %utilty of Circular Error Prob.   utils
%  inspace(i).StandoffUtility;             %utilty of Standoff Distance      utils
%  inspace(i).NumberUtility;               %utilty of Number in FB22         utils
%  inspace.MultiAttributeUtility;          %multi attribute utility          utils
% %-------------------------------------------------------------------------------------------

% Load Utility Data:
load udata_fake.mat;                          %data from MIST interviews        various

% Put the data into its respective columns:
x_Standoff = udata(:,1);    y_Standoff = udata(:,2);
x_Sorties = udata(:,3);    y_Sorties = udata(:,4);

%start the calculation:
%function [u] = utility(big_k,k_vector,attributes);
   % Load inputs:
   Standoff = [inspace.GlideDistance];       %Standoff Distance              nmi
   Sorties = [inspace.Sorties];              %# of sorties required to destroy target set

   % INTERPOLATE TO FIND SINGLE ATTRIBUUTE UTILITY VALUES:
      StandoffUtility = interp1(x_Standoff,y_Standoff,Standoff,'linear','extrap');
      SortiesUtility = interp1(x_Sorties,y_Sorties,Sorties,'linear','extrap');
```

```
   % Calculate Multi Attribute Utility:
   BigK = Constants.BigK;
   le_uns = ones(1,length(inspace));
   compound_product =                (BigK*Constants.Smallk(1)*StandoffUtility+le_uns);
   compound_product =
compound_product.*(BigK*Constants.Smallk(2)*SortiesUtility+le_uns);
   MultiAttributeUtility = (compound_product - le_uns)/BigK;
   for i = 1:length(inspace)
      inspace(i).StandoffUtility       = StandoffUtility(i);
      inspace(i).SortiesUtility        = SortiesUtility(i);
      inspace(i).MultiAttributeUtility = MultiAttributeUtility(i);
   end;
[outspace] = inspace;
```

## 2. Second Spiral

Since most of the code in the second spiral is reuse of the first, only the new module is presented here. Of course, there are small changes in the main file, constants file and the utility files, but they are obvious.

### a) SDBRetarget

```
%--------------------------------------------------------------------------------
% SDB Retarget
% This file assigns retarget time based on configuration.
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   DesignVector struct
% Outputs:
%   Retarget, a vector, is added to the Tradespace struct
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      if DesignVector(i).Actuators==2
         Tradespace(i).Retarget=4.0;
      else
         Tradespace(i).Retarget=2.6;
      end
   else
      if DesignVector(i).Actuators==2
         Tradespace(i).Retarget=4.6;
      else
```

```
        Tradespace(i).Retarget=3;
      end
   end
end
```

## 3. Third Spiral

Since most of the code is changed slightly for the third spiral, all twelve modules are presented here.

### a) SDBMain

```
%--------------------------------------------------------------------------------
% SDB Main
% This file calls all the other files in the correct order
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   none
% Outputs:
%   none
%--------------------------------------------------------------------------------

clear

SDBConstants                    %Calls constants into workspace
SDBDesignVectorOld              %Calls the DesignVector into workspace
SDBGlideDistance                %Calculates glide distance and L/D
SDBCrossSectionalDensity        %Calculates X-sectional density and length of bomb
SDBCircularErrorProbable        %Calculates Circular Error Probable
SDBNumberInFB22                 %Calculates how many SDBs will fit into an FB-22
SDBCostSchedule                 %Estimates Cost and Schedule of bomb--should be made
relative cost and schedule
[Tradespace] = SDBSortie(Tradespace,Constants,DesignVector); %Calculates number of sorties
necessary to destroy target set
SDBRetarget                     %Assigns retarget time based on configuration
SDBLoiter                       %Determines Max Loiter Time
[Tradespace] = SDBUtility(Tradespace,Constants);    %Calculates Utility
```

### b) SDBConstants

```
%--------------------------------------------------------------------------------
% SDB Constants
% This file initializes needed constants
% Author: Jason Derleth
% Date: 8/21/2002
```

```
% Version: 0.1
% Inputs:
%   none
% Outputs:
%   Constants structure
%--------------------------------------------------------------------------------

Constants.AverageBombDensity = 0.07;        %increased by 0.01 to account for fuel
Constants.LaunchSpeed = 750;                %In MPH
Constants.LaunchAltitude = 20000;           %In feet
Constants.DragCoefficientSub0 = 0.0045;     %Estimation from p.278 of Raymer 1st ed.
Constants.ProductionNumber=5000;            %WAG on number to be produced in 5 years
Constants.TestBombs=10;                     %WAG on number of flight test bombs to be used
Constants.LayersInBay=2;                    %in layers of bombs
Constants.BombBayWidth=30;                  %in inches
Constants.BombBayLength=18*12;              %in inches
Constants.NumBomBays=2;
Constants.NumAttributes=3;
%Constants.Smallk=[.275 .375 .475 .300];     %input from Excel sheet
Constants.Smallk=[.3 .7 .5 .4];             %input from Excel sheet
Constants.BigK=nirav_calculate_K(Constants.Smallk);   %Calculates Big K
Constants.BunkerKillDistance=1;             %in meters
Constants.TankKillDistance=3;               %in meters
Constants.APVKillDistance=4;                %in meters
Constants.JeepKillDistance=5;               %in meters
Constants.NumberOfBunkers=5;
Constants.NumberOfTanks=2;
Constants.NumberOfAPVs=5;
Constants.NumberOfJeeps=10;
Constants.AssumedKillPercent=0.95;          %Percentage where kill is assumed
Constants.SpecificFuelConsumption=0.8;      %pounds per hour per pound, estimated from table
in Raymer
Constants.MaxThrust=30;                     %pounds of thrust; estimated from low LOCAAS number
Constants.MaxMach=0.99;                     %entire model is subsonic
Constants.TurbineInletTemperature=70;       %Degrees F
```

**c) SDBDesignVector**

```
%--------------------------------------------------------------------------------
% DesignVector Enumeration
% This file enumerates the DesignVector for the Small Diameter Bomb Code
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.1
% Inputs:
%   None
% Outputs:
```

```
%   DesignVector struct:
%      DesignVector(i).AspectRatio          Ratio used to calculate lift, drag, glide distance...
%      DesignVector(i).Guidance             Either Inertial, GPS, or Active
%      DesignVector(i).ExplosiveWeight      In pounds of explosive
%      DesignVector(i).CasingWeight         In pounds, % of Explosive Weight
%      DesignVector(i).Diameter             In inches
%      DesignVector(i).FuelWeight           In Pounds
%--------------------------------------------------------------------------------

clear DesignVector
i=1;
for AspectRatio=[0.1:0.05:2]
   for Guidance={'inertial' 'GPS'}
      for Actuators=[2:3]
         for ExplosiveWeight=[150:10:220]
            for FuelWeight=[10:5:25]
               for Diameter=[3:1:9]
                  DesignVector(i).AspectRatio=AspectRatio;
                  DesignVector(i).Guidance=Guidance;
                  DesignVector(i).Actuators=Actuators;
                  DesignVector(i).ExplosiveWeight=ExplosiveWeight;

DesignVector(i).CasingWeight=(ExplosiveWeight+FuelWeight)*(AspectRatio*10)*(Diameter/1
2)*.1;
                  DesignVector(i).Diameter=Diameter;
                  DesignVector(i).FuelWeight=FuelWeight;
                  i=i+1;
               end
            end
         end
      end
   end
end
```

**d) SDBGlideDistance**

```
%--------------------------------------------------------------------------------
% SDB Glide Distance Calculation
% This file calculates the Glide Distance
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   DesignVector struct
%   Tradespace.CrossSectionalDensity struct
% Outputs:
%   Tradespace.GlideDistance array added on to Tradespace struct, all distances in mi
```

```
%   Tradespace.LiftOverDrag vector added to Tradespace struct
%   Tradespace.MaxFlightTime vector added to Tradespace struct
%--------------------------------------------------------------------------------


for i=[1:size(DesignVector,2)];

   % First, we must calculate e, the Oswald Span Efficiency:
   e=1.78 * (1-0.045 * (DesignVector(i).AspectRatio ^ 0.68)) - 0.64; % From p.297 of Raymer
1st ed.

   %Now, we calculate the Glide Ratio, i.e., how much do we drop in z for every distance in x:
   GlideRatio=0.5 * sqrt((pi * DesignVector(i).AspectRatio * e) /
Constants.DragCoefficientSub0); % From p.467 of Raymer 1st ed.

   %Given that and an initial height, we know how far the bomb will glide:
   Tradespace(i).GlideDistance = [(GlideRatio * Constants.LaunchAltitude) / 6076.115];  % in
nautical miles
   Tradespace(i).LiftOverDrag = GlideRatio;
end
```

**e) SDBCrossSectionalDensity**

```
%--------------------------------------------------------------------------------
% SDB Cross-Sectional Density Calculation
% This file calculates the Cross-Sectional Density for the Small Diameter Bomb Code
%          and also determines the length of the bomb.
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.0
% Inputs:
%   DesignVector struct
% Outputs:
%   Tradespace.CrossSectionalDensity
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
   Tradespace(i).CrossSectionalDensity=[(DesignVector(i).CasingWeight +
DesignVector(i).FuelWeight + ...
         DesignVector(i).ExplosiveWeight) / DesignVector(i).Diameter];
      % CrossSectionalDensity in lbs/diameter
   Tradespace(i).Length=[(DesignVector(i).CasingWeight + DesignVector(i).ExplosiveWeight +
...
         DesignVector(i).FuelWeight) / (Constants.AverageBombDensity * pi *
((DesignVector(i).Diameter / 2)^2))];
      % Length in inches
   Tradespace(i).Diameter=DesignVector(i).Diameter; %in inches
```

```
%    Tradespace(i).Wingspan=(Tradespace(i).Length)/24; %in feet, wings assumed to be 1/4 the
length of the bomb each
end
```

**f) SDBCircularErrorProbable**

```
%--------------------------------------------------------------------------------
% SDB Circular Error Probable
% This file calculates the distance the bomb can fly under powered flight.
%       It does so using the Breguet Range Equation for jets, p. 456 Raymer 1st ed.
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   Constants.SpecificFuelConsumption
% Outputs:
%   Adds Circular Error Probable, a vector, to the Tradespace struct
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      Tradespace(i).CircularErrorProbable = 13-2*DesignVector(i).Actuators;
   else
      Tradespace(i).CircularErrorProbable = 8-2*DesignVector(i).Actuators;
   end
end
```

**g) SDBNumberInFA22**

```
%--------------------------------------------------------------------------------
% SDBNumberInFA22.m
% This file calculates the number of SDBs that will fit into an F/A-22
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 1.0
% Inputs:
%   DesignVector struct
%   Constants.BombBayHeight
%   Constants.BombBayWidth
%   Constants.BombBayDepth
% Outputs:
%   Tradespace.NumberInFA22 vector added to Tradespace Struct
%--------------------------------------------------------------------------------

Margin=0.1; %ten percent margin on width and depth to account for mounting system

for i=[1:size(DesignVector,2)];
```

```
   HeightNum=Constants.LayersInBay; % LayersInBay is in number of layers of bombs that fit
in bay
   WidthNum=round((Constants.BombBayWidth*(1-Margin)/(DesignVector(i).Diameter))-0.5);
   LengthNum=round((Constants.BombBayLength*(1-Margin)/(Tradespace(i).Length))-0.5);

Tradespace(i).NumberInFA22=HeightNum*WidthNum*LengthNum*Constants.NumBomBays;
end
```

**h) SDBSortie**

```
function [outspace] = SDBSortie(Tradespace, Constants, DesignVector);
%--------------------------------------------------------------------------------
% SDB Sortie counter
% This file calculates the number of sorties needed to destroy the chosen target set.
% Author: Jason Derleth
% Date: 3/18/03
% Version: 1.0
% Inputs:
%   DesignVector struct
%   Tradespace Struct
% Outputs:
%   Tradespace.Sorties
%--------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];

   if Tradespace(i).NumberInFB22==0
      Tradespace(i).Sorties=0;
   else
      ScalingFactor=sqrt(DesignVector(i).ExplosiveWeight/220);    %Divided by max explosive
weight
      BunkerKill=Constants.BunkerKillDistance*ScalingFactor;
      TankKill=Constants.TankKillDistance*ScalingFactor;
      APVKill=Constants.APVKillDistance*ScalingFactor;
      JeepKill=Constants.JeepKillDistance*ScalingFactor;
      sigma=Tradespace(i).CircularErrorProbable/0.68;          %Gives us sigma for normal
distribution
                                          %i.e., 50% of bombs fall within given CEP

      x=[-BunkerKill BunkerKill];
      P=normcdf([-BunkerKill BunkerKill],0,sigma);
      NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
         log(1-(1-(P(2)-P(1)))^2));
      NumPairs=ceil(NumPairs);
      NumBombs=2*NumPairs*Constants.NumberOfBunkers;
```

```
        x=[-TankKill TankKill];
        P=normcdf([-TankKill TankKill],0,sigma);
        NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
            log(1-(1-(P(2)-P(1)))^2));                  %guarantee assumed kill percentage
        NumPairs=round(NumPairs+0.5);
        NumBombs=NumBombs+2*NumPairs*Constants.NumberOfTanks;

        x=[-APVKill APVKill];
        P=normcdf([-APVKill APVKill],0,sigma);
        NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
            log(1-(1-(P(2)-P(1)))^2));                  %guarantee assumed kill percentage
        NumPairs=round(NumPairs+0.5);
        NumBombs=NumBombs+2*NumPairs*Constants.NumberOfAPVs;

        x=[-JeepKill JeepKill];
        P=normcdf([-JeepKill JeepKill],0,sigma);
        NumPairs=(2*(log(Constants.AssumedKillPercent))/...  %# of bombs needed when dropped
in pairs to
            log(1-(1-(P(2)-P(1)))^2));                  %guarantee assumed kill percentage
        NumPairs=round(NumPairs+0.5);
        NumBombs=NumBombs+2*NumPairs*Constants.NumberOfJeeps;


        Sorties=(NumBombs/Tradespace(i).NumberInFB22);
        Tradespace(i).Sorties=round(Sorties+0.5);
        Tradespace(i).NumBombs=NumBombs;

    End
end
[outspace] = Tradespace;
```

**i) SDBRetarget**

```
%-------------------------------------------------------------------------------
% SDB Retarget
% This file assigns retarget time based on configuration.
% Author: Jason Derleth
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%
% Outputs:
%
%-------------------------------------------------------------------------------
```

```
for i=[1:size(DesignVector,2)];
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      if DesignVector(i).Actuators==2
         Tradespace(i).Retarget=4.0;
      else
         Tradespace(i).Retarget=2.6;
      end
   else
      if DesignVector(i).Actuators==2
         Tradespace(i).Retarget=4.6;
      else
         Tradespace(i).Retarget=3;
      end
   end
end
```

**j) SDBLoiter**

```
%-------------------------------------------------------------------------------
% SDB Loiter Time Calculation
% This file calculates loiter time of the bomb
% Author: Jason Derleth
% Date: 4/14/2003
% Version: 1.0
% Inputs:
%   DesignVector struct
% Outputs:
%   Tradespace.Loiter
%-------------------------------------------------------------------------------

for i=[1:size(DesignVector,2)];

InitialWeight=DesignVector(i).CasingWeight+DesignVector(i).FuelWeight+DesignVector(i).Ex
plosiveWeight;
   FinalWeight=DesignVector(i).CasingWeight+DesignVector(i).ExplosiveWeight;
   EnduranceLiftOverDrag=0.866*Tradespace(i).LiftOverDrag;
   Tradespace(i).Loiter=60*(EnduranceLiftOverDrag/Constants.SpecificFuelConsumption)*...
      log(InitialWeight/FinalWeight);        %Loiter Time in minutes
end
```

**k) SDBCostSchedule**

```
%-------------------------------------------------------------------------------
% SDB Cost and Schedule
% This file calculates the distance the bomb can fly under powered flight.
%       It does so using the Breguet Range Equation for jets, p. 456 Raymer 1st ed.
% Author: Jason Derleth
```

```
% Date: 8/13/2002
% Version: 0.1
% Inputs:
%   Constants, Tradespace, DesignVector structures
% Outputs:
%   Adds TotalCost and TotalHours, vectors, to the Tradespace struct
%--------------------------------------------------------------------------------

EngRate=86;
ToolRate=88;
QCRate=81;
MfgRate=73;

 for i=[1:size(DesignVector,2)];
   TotalWeight=DesignVector(i).CasingWeight + DesignVector(i).ExplosiveWeight;

EngHours=(7.07*(TotalWeight)^0.777)*(Constants.LaunchSpeed^0.894)*(Constants.Production
Number^0.163);

ToolHours=(8.71*(TotalWeight)^0.777)*(Constants.LaunchSpeed^0.696)*(Constants.Productio
nNumber^0.263);

MfgHours=(10.72*(TotalWeight)^.82)*(Constants.LaunchSpeed^0.484)*(Constants.Production
Number^0.641);
   QCHours=0.133*MfgHours;
   DevelopSupportCost=(7.07*(TotalWeight)^0.630)*(Constants.LaunchSpeed^1.3);

TestCost=(1807.1*(TotalWeight)^0.325)*(Constants.LaunchSpeed^0.822)*(Constants.TestBom
bs^1.21);

MfgMaterialsCost=(16*(TotalWeight)^0.921)*(Constants.LaunchSpeed^0.621)*(Constants.Prod
uctionNumber^0.799);
   Tradespace(i).TotalHours=EngHours + ToolHours + MfgHours + QCHours;
   EngCost=2251*(0.043*Constants.MaxThrust + 243.25*Constants.MaxMach +
0.969*Constants.TurbineInletTemperature);
   temp=strcmp(DesignVector(i).Guidance,'inertial');
   if temp>0
      GuidanceCost=4000*DesignVector(i).Actuators;
   else
      GuidanceCost=3000+4000*DesignVector(i).Actuators;
   end
   BombCost=(EngHours*EngRate + ToolHours * ToolRate + MfgHours * MfgRate + QCHours
* QCRate + DevelopSupportCost + TestCost + MfgMaterialsCost);
   Tradespace(i).TotalCost=BombCost+GuidanceCost;
end
```

**l) SDBUtility**

```
function [outspace] = SDBUtility(inspace,Constants);
%-----------------------FUNCTION INPUTS-------------------------------------------------
% Constants.BigK;                          %multiplicative constant          unitless
% Constants.Smallk;                        %single attribute constants       unitless
% CEP = inspace(i).CircularErrorProabable;   %Circular Error Probable          meters
% Standoff = inspace(i).GlideDistance;       %Standoff Distance               nmi
% Num = inspace(i).NumberInFB22;             %# That fit in FB-22              bombs
%---------------------------------------------------------------------------------------


% %----------------------FUNCTION OUTPUTS-----------------------------------------------
%  inspace(i).CEPUtility;                  %utilty of Circular Error Prob.   utils
%  inspace(i).StandoffUtility;             %utilty of Standoff Distance      utils
%  inspace(i).NumberUtility;               %utilty of Number in FB22         utils
%  inspace.MultiAttributeUtility;          %multi attribute utility          utils
% %---------------------------------------------------------------------------------------

% Load Utility Data:
load udata_fake.mat;                          %data from MIST interviews        various


% Put the data into its respective columns:
x_Standoff = udata(:,1);    y_Standoff = udata(:,2);
x_Sorties = udata(:,3);     y_Sorties = udata(:,4);
x_Retarget = udata(:,5);    y_Retarget = udata(:,6);
x_Loiter = udata(:,7);      y_Loiter = udata(:,8);



%start the calculation:

%function [u] = utility(big_k,k_vector,attributes);

   % Load inputs:
   Standoff = [inspace.GlideDistance];       %Standoff Distance        nmi
   Sorties = [inspace.Sorties];              %# of sorties required to destroy target set
   Retarget = [inspace.Retarget];            %Time before impact retarget will work   seconds
   Loiter = [inspace.Loiter];                %Loiter time in hours


   % INTERPOLATE TO FIND SINGLE ATTRIBUUTE UTILITY VALUES:
      StandoffUtility = interp1(x_Standoff,y_Standoff,Standoff,'linear','extrap');
      SortiesUtility = interp1(x_Sorties,y_Sorties,Sorties,'linear','extrap');
      RetargetUtility = interp1(x_Retarget,y_Retarget,Retarget,'linear','extrap');
      LoiterUtility = interp1(x_Loiter,y_Loiter,Loiter,'linear','extrap');
```

```
inspace;
   % Calculate Multi Attribute Utility:
   BigK = Constants.BigK;
   le_uns = ones(1,length(inspace));
   compound_product =                (BigK*Constants.Smallk(1)*StandoffUtility+le_uns);

   compound_product =
compound_product.*(BigK*Constants.Smallk(2)*SortiesUtility+le_uns);
   compound_product =
compound_product.*(BigK*Constants.Smallk(3)*RetargetUtility+le_uns);
   compound_product = compound_product.*(BigK*Constants.Smallk(4)*LoiterUtility+le_uns);

   MultiAttributeUtility = (compound_product - le_uns)/BigK;

   for i = 1:length(inspace)
      inspace(i).StandoffUtility      = StandoffUtility(i);
      inspace(i).SortiesUtility       = SortiesUtility(i);
      inspace(i).RetargetUtility      = RetargetUtility(i);
      inspace(i).MultiAttributeUtility = MultiAttributeUtility(i);
      inspace(i).LoiterUtility        = LoiterUtility(i);
   end;
[outspace] = inspace;
```

## B. Appendix: Bibliography


Air Force Guide to Evolutionary Acquisition, draft of November 2000, SAF/AQ Evolutionary Acquisition Reinvention Team

Air Force Instruction 63-123, April 2000

Ang, Alfredo H-S and Wilson H. Tang, Probability Concepts in Engineering Planning and Design, New York, John Wiley & Sons, 1975

De Neufville, Richard and David H. Marks: Systems Planning and Design, Cambridge, MA: MIT Dept. of Civil Engineering 1973

Diller, Nathan: Utilizing Multiple Attribute Tradespace Exploration with Concurrent Design for Creating Aerospace Systems Requirements, MIT Thesis 2002

Fortescue, Stark, and Swinerd editors: SpaceCraft Systems Engineering, Hoboken, NJ: J. Wiley, 2003

Highsmith III, James A.: Adaptive Development, a Collaborative Approach to Managing Complex Systems, New York, Dorset House Pub., 2000

Keeney, Ralph L. and Howard Raiffa, Decisions with Multiple Objectives, Cambridge, UK, Cambridge University Press, 1993

Maier, Mark and Eberhardt Rechtin: The Art of Systems Architecting, Boca Raton, CRC Press, 2000.

Raymer, Daniel P. Aircraft Design: A Conceptual Approach, 3rd. ed. Reston, VA: American Institute of Aeronautics and Astronautics, inc, 1999

Ross, Adam: Multi-Attribute Tradespace Exploration with Concurrent Design as a Value-Centric Framework for Space System Architecture and Design, MIT Thesis 2003

Zarchan, Paul: Tactical and Strategic Missile Guidance, Reston, VA: American Institute of Aeronautics and Astronautics 2002