Spring 5-2017

Determining Feasibility Resilience: Set Based Design Iteration Evaluation Through Permutation Stability Analysis

James E. Ross
University of Southern Mississippi

DETERMINING FEASIBILITY RESILIENCE: SET BASED DESIGN ITERATION EVALUATION THROUGH PERMUTATION STABILITY ANALYSIS

by

James E. Ross

A Dissertation
Submitted to the Graduate School
and the School of Computing
at The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved:

Dr. Zheng Sun, Committee Chair
Associate Professor, Computing

Dr. Andrew Strelzoff, Committee Member
Senior Research Scientist, Army Corps of Engineers

Dr. Chaoyang Zhang, Committee Member
Professor, Computing

Dr. Zheng Wang, Committee Member
Assistant Professor, Computing

Dr. Nan Wang, Committee Member
Associate Professor, Computing

Dr. Andrew Sung
Director, School of Computing

Dr. Karen S. Coats
Dean of the Graduate School

May 2017

ABSTRACT

DETERMINING FEASIBILITY RESILIENCE: SET BASED DESIGN ITERATION EVALUATION THROUGH PERMUTATION STABILITY ANALYSIS

by James E. Ross

May 2017

The goal of robust design is to select a design that will still perform satisfactorily even with unexpected variation in design parameters. A resilient design will accommodate unanticipated future system requirements. Through studying the variations of system parameters through the use of multi-objective optimization, a designer hopes to locate a robustly resilient design, which performs current mission well even with varying system parameters and is able to be easily repurposed to new missions. This ability to withstand changes is critical because it is common for the product of a design to undergo changes throughout its life cycle. This subject has been an active area of research in industrial design and systems engineering but most methodologies rest upon exhaustive understanding of design, manufacturing and mission variance. The thrust of this research is to develop new methodologies for estimating robust resilience given imperfect information. In this work we will apply new methodologies for locating resilient designs within a dataset derive from a study performed by the Small Surface Combatant Task Force in order to improve upon a state of the art design process. Two new methodologies, permutation stability analysis and mutation stability analysis, are presented along with results and discussion as applied to the SSCTF dataset. It is demonstrated that these new methods improve upon the state of the art by providing insight into the robustness and resilience of selected system properties. These methodologies, although applied to the SSCTF dataset are posed more generally for wider application in system design.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my committee chair Dr. Jonathan Sun. Dr. Sun kept me motivated and moving forward as I have a tendency to spend too much time on the details. His persistence and encouragement in regards to academic research motivated me to keep moving forward and to meet my deadlines on time. Without our Friday lunch meetings, and Dr. Sun's dedication to keeping me on track, this work would not have been completed.

I would also like to give special recognition to Dr. Andrew Strelzoff. His ability to identify relevant motivating ideas and to recognize the impact of those ideas was the inspiration for this work. His tirelessly patient efforts into my academic career will always hold a special memory for me.

I would like to thank committee members Dr. Chaoyang Zhang, Dr. Zheng Wang, and Dr. Nan Wang for their time and encouragement in the pursuit of academic excellence. Time is the greatest gift anyone can give anyone else, and for their time, I thank them. In addition, I would like to thank ERDC: Information Technology Laboratory for their dedication to academic pursuits, and providing all of the tools necessary for completing my research. Without their encouragement and support, this work would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ABBREVIATIONS

AMPV	Armored Multi-Purpose Ground Vehicle
C3	Command and Control Centers
LCS	Littoral Combat Ship
MOO	Multi Objective Optimization
SBD	Set Based Design Methods
SME	Subject Matter Experts
SSCTF	Small Surface Combatant Task Force

CHAPTER I - INTRODUCTION

The Small Surface Combatant Task Force (SSCTF) was created in 2014 with the purpose of examining existing Littoral Combat Ship (LCS) designs, modified LCS designs, and new design concepts. With these designs, the goal was to determine if the existing LCS ship could be modified to meet today and future mission needs, or if a completely new design would be a better option. In order to be able to answer the question of whether to buy a new ship or to modify the existing model, the Navy began its design process. The naval design process is a complicated time-consuming process that requires the skill set of a group of highly specialized naval architects. The group of naval architects is required to meet many times in order to finalize a ship design. The process of meeting and redesign requires weeks of time and many meetings. Steps towards a final design are small and potential changes to the design always left more designs to be created. All designs created were to be placed into a tradespace made up of possible designs in order to ensure a large number of designs were analyzed. The designs were to include the lethality of the ship towards land, sea, and air. The designs were also to include cost, combat systems, and weapons. The goal of the SSCTF was to determine whether or not to purchase a new Navy vessel or to continue to use the existing LCS to fulfill mission requirements through a lengthy study of the capabilities of the current LCS against the expected capabilities of a new ship design.

Part of the goal of the Navy ship design team was to develop a more robust tradespace than had previously been possible given time and financial restraints on the ship design process. In order to develop this more robust tradespace, The Small Surface Combatant Task Force (SSCTF) used Set Based Design Methods (SBD). The SBD method was followed using steps defined in the technical paper titled “What is Set-Based Design” (David J. Singer, Doerry, and Buckley 2009). The first step in the process is to define bounds for the tradespace so that designs are not created outside of the tradespace area of interest. Next, the designer must ensure that the tradespace is sufficiently large enough to fulfill the density of designs requirement as determined by the design expert. Once a sufficiently large tradespace has been created, then the tradespace should be analyzed by subject matter experts focusing on the design alternatives within their domain of specialty. During the analyses, the design experts should eliminate designs that will not produce a good solution. A good solution is a design that is capable of fulfilling the requirements of the design and is also known as a feasible design. During this reduction in the tradespace, designs that are feasible will emerge and a tradespace with more viable options to the designers will be created. Once designers have ensured that the tradespace is sufficiently large and eliminated designs in the tradespace in their domain of specialty, the remaining designs from each group of specialist should be recombined into one tradespace. The end result of this process of populating a tradespace and then reducing the size of the tradespace by design experts will result in a tradespace that is more robust than had previously been possible by allowing more feasible ship designs to be considered in the final design options.

The Navy ship design study was focused on creating a robust tradespace of possible designs for each of the 5 Hull Mechanical and engineering (HM&E) configurations. These 5 configurations are called the design Seeds and represent the 5 different propulsion systems studied during this effort. The 5 Seeds are as follows:

- Mechanical Drive Twin Shaft
- Mechanical Drive Single Shaft
- Integrated Power System twin Shaft
- Integrated Power System Single Shaft
- Integrated Power System twin Shaft, Adjacent motors

The 5 different HM&E configurations are also known as M1, M2, I1, I2, IC. M1 designates the single propeller propulsion system. M2 is the dual propeller based system in which the propellers are not located in the same compartment within the ship. I1 is the single electrical propulsion system. I2 is the dual electrical propulsion system in which the propulsion systems are not dual located (located within the same compartment). IC is a dual electrical based system in which the electrical propulsion systems are located within the same compartment. The mechanical propulsion systems have been around for enough years to establish a wealth of historical data leading to a better understanding of the limitations and also the benefits of using such a system. The electrical propulsion systems are very new in comparison to its mechanical alternatives. The electrical propulsion systems have a large amount of potential but until they have had more years of use, the limitations and benefits are not be fully understood. In addition, the cost associated with using a mechanical based propulsion system is much lower than using an Integrated Power System so even though the Mechanical system wins out in the cost category, it produces less power for use in current and future components of the ship than the Integrated Power System. Even if the Mechanical System is able to handle the power requirements of the current ship design, it may not be able to handle future power requirements of the ship. Even though Integrated Power Systems have not been around for a long time and there is not a large amount of information available, Integrated Power Systems may enable use of electronics that might require more electricity than what current mechanical systems produce.

When choosing the best mechanical model of a naval ship, it is helpful to use the ship base configuration with the most feasible designs in order to ensure that changes made to a design do not cause a ship to become infeasible. It will be helpful to identify which ship has the most feasible designs. In order to identify which ship has the most feasible designs, we are going to look at 5 different mechanical models of ship design, and we are also going to look at the number of feasible designs that are affected by 4 critical variables of the ship design.

The four critical variables of ship design are Free Power, Free Weight, Free Cost, Free Space. Power is the variable responsible for holding a value representing the amount of power the ship is capable of maintaining. Weight is the amount of weight the ship can hold without becoming unstable and sinking. Cooling is the amount of Cooling available for ship components. Space is the amount of space remaining on the ship after the expected components have already been added to the ship. These 4 variables have the biggest overall impact on ship feasibility so making good choices for the values of these 4 parameters for each of the 5 HM&E configurations will likely result in a good design.

The robust trade space created by the SSCTF was based upon the idea of Capability Concepts. “A Capability Concept is a set of operational capability levels and an associated CONOPS for employing the capabilities” (Garner et al. 2015). An example would be the capability of the design handling itself versus submarines in relation to the extent it is capable of offense or defense. An example of a chart representing operational capability levels of a design can be seen in Figure 1 (Garner et al. 2015). As one moves further out from the center of the chart in each of the 14 example operational capabilities, the overall operational capability of the design increases for the Capability Concept that this design represents. A completed bullseye chart displaying all Operational Capabilities for a Capability Concept is called a Configuration. An example of a configuration is represented by Figure 1. There can be many configurations that meet all the requirements of a Capability Concept, which means the Capability Concept is a feasible concept and meets the "current level of fidelity and analysis" (Garner et al. 2015). There can also be Capability Concepts in which no configurations meet all the requirements. These capability concepts are infeasible thus do not meet current fidelity requirements.

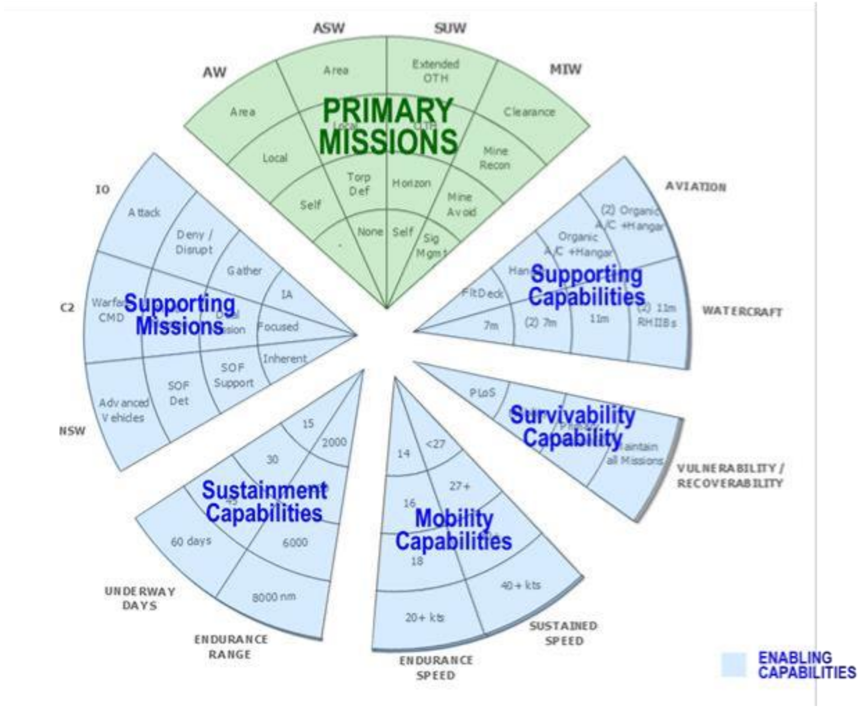


Figure 1. Operational capability levels in a Bulls eye chart

(Garner et al. 2015)

For the example study performed in this work, the list of initial Capability Concepts began with 192 different Capability Concepts. This list was reduced to 13 different Capability Concepts and then further to 8 Capability Concepts. The reduced set of Capability Concepts can be viewed within the bullseye chart in Figure 1 along with a few of the Capability Concepts that were eliminated from final consideration. The elimination of Capability Concepts was performed by area experts and is not covered in this material. For the 8 remaining Capability Concepts, there are “mission system alternatives (MAs) designed to achieve a complete detect-to-engage capability for a mission area capability level” (Garner et al. 2015). An example of a MA would be the ship’s ability to perform all tasks required from detecting to engaging in warfare with an aircraft. Using different MAs for the four primary mission capabilities seen in figure 1, over 2000 different Combat Capability Alternatives were created. Then estimates for 4 the primary variables, Power, Space, Weight, and Cooling were developed. An example of Combat Capabilities and how they relate to MAs can be seen in Table 1 (Garner et al. 2015).

Table 1

Example of Mission Area Capabilities and Capability Concepts

Capability Concept					
Mission Area Capabilities	CC 1	CC 2	CC 3	CC 4	CC 5
Self Defense against Air, Surface, Undersea Threats	X		X	X	
Capability to detect and engage small craft within- the- horizon of own ship		X	X	X	X
Capability to achieve mission kill of			X		X

over-the-horizon surface targets					

(Garner et al. 2015)

Even if a Design is found to be feasible, it may not be a viable design. Some designs will pass the requirements of a Capability Concept but will not pass future testing and analysis (Garner et al. 2015). For example, a ship design with engine A may fill all the Operational Capabilities of a Capability Concept today but may fail to satisfy those same Operational Capabilities at a future time. As Time moves forward after a ship is built, it always gets heavier as more components are added to the ship. The ship's engine may have been capable of maintaining a level of speed but after the ship has become heavier than that same engine will no longer be able to maintain that level of speed. With the problem of not all designs that are found to be feasible remaining viable throughout the design's lifespan, finding ship designs that will pass future testing and analysis is critical.

When examining a variable of interest in a Capability Concept, it is helpful to assign a combined score to all the configurations the Capability Concept. Assigning a combined score to all the configurations of the Capability concept is a better option than relying on any one design in a configuration because even if a design is viable now does not mean it will be viable in the future. According to Garner, a "diverse group of configurations will mitigate "the risk that any one configuration will prove not viable" (Garner et al. 2015).

In order to allow a wider range of values to be studied for each ship design, sets of regression equations were developed for each of the 5 seeds using the statistical software JMP. The regression equations were valuable in allowing approximations for of configuration properties of specific “Rapid Ship Design Environment or RSDE Configurations”. RSDE is the tool that was used for creating a table of configurations representing the data space of designs (Garner et al. 2015).

The Engineering Resilient Systems (ERS) tradespace Toolkit was used to combine “regression equations, the cost algorithms, HM&E crew size algorithms, other algorithms and the data associated with the CCAs” (Garner et al. 2015). By combining the pieces of information into one software package, the ERS team was able to assist in the ship design process. The ERS team was responsible for generating the estimates using Monte Carlo methods. The final result of the ERS tradespace tool was the generation of approximately 10000 feasible designs for each of the 2000+ CCAs. After generation of the feasible designs produced by the ERS Tradspace Toolkit, some of the designs were compared with existing Small Surface Combatant designs and designs produced by the Small Surface Combatant team in order to determine the validity of the ERS tradespace toolkit results. It was determined that the results of the ERS Tradespace Toolkit were valid and could be used for producing possible designs.

After the ERS tradespace tool finished generating data, the Feasibility Element Calculator was to determine the feasibility of each design the was produced by the ERS tradespace tool. The 4 levels of feasibility are Feasible Excessive, Feasible, High Risk for Feasibility, and Not Feasible (Garner et al. 2015). Feasible Excessive represents a ship design that far exceeds the requirements of the desired ship design which sounds good, but the design is likely to be very high in cost. A feasible design is one that fills all requirements of the desired design but does not overly exceed the design requirements, thus being the most ideal level of design. High Risk For Feasibility means that it is unlikely that the ship design will be able to meet all the requirements of the desired ship design. Low Feasibility implies that the requirements of the desired ship design will not be met by this design. Each of the elements found in Table 2 are assigned a feasibility score. A design can also be found infeasible if more than 5 of the elements in table 2 are found to be High Risk For Feasibility.

Table 2

Feasibility Elements of Ship Design

SUW Performance
ASW Performance
Sea keeping
AW Performance
Sustained Speed
Endurance Speed
Arrangeable Area
Displacement
Length to Beam Ratio
Stack Up Length

(Garner et al. 2015).

During the stage of generating ship designs using the ERS Tradespace Toolkit, it was noticed that it would be very helpful to identify designs that were resistant to failure if changes were made to the CCA. A design that was resistant to failure if changes were made to the CCA would be a resilient design. Even if the design was not the most optimal design in the set of designs, if the design is more resistant to failure than the optimal design, it would be a better option. Noticing this need for an understanding of resilience in tradespace analysis was the source of inspiration for this paper.

CHAPTER II – BACKGROUND: INTRODUCTION TO RESEARCH

The problem with exploring the full space of design options prior to any narrowing is the complexity of determining complex interrelationships and tradeoffs among design parameters. Multi-Objective Optimization (MOO) is a way to help analyze the complex relationships between design parameters. MOO is the process of attempting to find an optimum balance between the parameters that make up a design. The parameters that make up a design are numerical value representations of the components of a design such as the expected power output from an engine or the diameter of a cylinder in an engine. In MOO problems designers are generally looking for the best possible design by balancing differing objectives. The problem with this approach is searching for the optimal design is often a time-consuming process; to make matters worse, the optimal design is often unable to withstand even the smallest of changes to its parameters. Often the resulting design from an MOO process works great but only under specific conditions; this leads to the desire to find an optimal design and a design that is also able to withstand changes.

A design that is able to withstand changes is known as a robust design. Since it is very difficult to locate a design that is completely robust, designs can be given a resiliency score that represents the measure of robustness. In order to help find a more resilient score, Set-based design can be utilized. Set-based design is a method of design in which areas of a design are analyzed in parallel. This parallel analysis allows each design team to focus on areas of the design without having to worry about how their design area affects other steps in the design process; design teams are able produce a more robust design as a result of set-based design. All design teams for a particular design are able to work on their area of the design concurrently. The result of this design method is that more time is spent searching the solution space of possible options for each area of the design, but a good option for each component in the design is available. Although set-based design is a newer concept in the design world it can be highly effective in ensuring that the final design chosen is among the set of best design options for a designer. Fundamentally, set-based design is about deferring design choices until the full space of possible designs has been fully explored (David J. Singer, Doerry, and Buckley 2009). In the early stages of the design process, narrowing design options based on incomplete or inadequate exploration leads to non-optimal solutions (David J. Singer, Doerry, and Buckley 2009). If choices are made but turn out to be poor choices then correcting these choices can often be both time consuming and costly (Vlahakis and Partridge 1989).

In this work, we will search for a design using Multi-Objective Optimization (MOO) techniques and of the designs found we are going to attempt to locate a set of designs that are robust. In our pursuit of the set of robust designs, we are going to look at examples of MOO and at methods used to generate, explore, and filter data for MOO. We are going to examine methods and concepts used for expressing uncertainty and Imprecision in the design process. We are going to explore strategies used in searching for an optimal design and concepts necessary for understanding how to recognize the solution space of optimal designs and how to search for a robust design using strategies of substitution and modification of parameter values. This substitution of parameter values will assist in the assignment a resiliency score, which will represent a design's ability to withstand changes. The resiliency score will allow us to identify a design that may not be the best overall design but will be likely to outperform the best design in terms of being able to adapt to predicted and unpredicted changes to the parameters representing the components of the design. By combining the concepts of multi-objective optimization, set-based design, and resiliency, this work hopes to both encourage ideas and develop new methods in locating sets of resilient design within a design space.

Introduction to Multi-Objective Optimization

In the early stages of the design process, choices are often made on incomplete or inadequate information. The problem with having to make choices in the early stages with partial information is that often these decisions are critical and will have long lasting impacts on the overall design. If choices are made that turn out to be poor choices then correcting those choices can often be both time consuming and costly. After modifications have been made to the initial design, eventually an acceptable design will be created which can include information on manufacturing imprecision. Imprecision is a known issue in manufacturing as it is highly unlikely to obtain two products that are identical. As Daum explains, when having cylinders manufactured, a designer can request two cylinders of 50mm and instead of receiving two cylinders of exactly 50mm, the designer will instead receive two cylinders within a predefined manufacturing range of 50mm (Correa Florez, Bolaños Ocampo, and Escobar Zuluaga 2014). The range is often small but the variations in final manufactured product are expected. Imprecision is closely related to the issue of uncertainty. Imprecision is how far from the intended specifications was the item that was produced; uncertainty is knowing that there is going to be some degree of difference between the final product and the designed product, but not knowing for sure the variation from the specifications of the product. As Antonsson describes, the tools designers use often do not have any way of capturing imperfections in the manufactured final product (Josephson et al. 1998). Because the designer's tools are often incapable of capturing imperfections in the manufactured product, a tool that could help with providing complete or adequate information to the designer would be very useful.

Multi-Objective Optimization is the perfect resource for designers to leverage to help with providing more information on a design before critical decisions are made. Multi-objective optimization is the process of attempting to maximize the effectiveness of a design by managing the objective function values of multiple objective functions. An objective function is a function that provides numerical representation to the parameters that make up a design. An objective function can utilize other objective functions to provide its numerical representation of a parameter. By helping to identify the preferred combinations of objective function values, MOO is able to provide designers with more information before critical design decisions are made. However, according to Fonseca, real world problems involving multi-objective optimization problems are often difficult to solve due to conflicting requirements of the objective function. The overall goal of multi-objective optimizations turns into a level of acceptance for the parameters of the objective function, which is the result of a compromise in value between the objective function parameters (Fonseca, Fleming, and others 1993). While MOO helps designers to locate optimal designs, it is important to realize that real work usage of MOO is likely to be a measure of acceptance as Fonseca described.

Introduction to Conflicting Objective Values

In many real world problems, it is a more common issue to have conflicting objectives than to not have conflicting objectives; Multi-Objective Optimization (MOO) deals with objective functions possessing conflicting objectives. Conflicting objectives are objective function values that require the designer to manage the output value of two or more objective functions that require a choice to be made where one objective function will benefit and the conflicting objective function(s) values will be penalized. The goal of Multi-Objective Optimization is to locate an ideal solution and thus a balance in gain versus loss for each of the objectives in the objective function must be found (M. T. M. Emmerich, Giannakoglou, and Naujoks 2006). An example would be, a designer could want a design that has 3 weapon systems but only have the budget to afford two weapon systems. The conflict between wanting more resources and the cost of the resources is an example of conflicting objectives.

Studies on Conflict

The study of multi-objective optimization ranges across many fields in the search of ways to handle conflictive objective function values. In the study performed by Llopis-Albert, the author described an efficient multi-objective algorithm for scheduling robot tasks such as trajectory planning and physical movement. The algorithm described takes into consideration collision avoidance and the time it takes to perform real-world tasks. The decisions made by the algorithm are based on a selection from a pareto-optimal frontier of possible choices representing "trade-offs between the different decision variables of the multi-objective optimization problem (Llopis-Albert, Rubio, and Valero 2015). A pareto-optimal frontier is the set of solutions that are considered Pareto efficient. A Pareto efficient solution is a solution in which making a change to any objective function value cannot improve the solution value. The result of the work by Llopis-Albert is an algorithm that helps show the trade-offs for choosing different decision variables and the time required for performing each task so that the time and cost efficiency of the robot’s tasklist is maximized. In another study performed by Kaitaniemi, the author studies the use of multi-objective optimization in order to determine ecological and evolutionary causes for changes in the life cycle of a specific species of moth. Normally when studying life cycles of insects, researchers tend to use single objective optimization but using a single objective for optimization is inefficient due to the life cycle of an insect requiring the contribution of many objectives (Kaitaniemi et al. 2012). Expansion of knowledge in multi-objective optimization is useful by increasing the understanding of a design solution space and will result in a better overall design of a final product or, in the case of insects, better understanding and prediction of life cycles. In a study by Chen who is also dealing with conflicting objective function values, Chen describes a method which seeks to identify the tradeoffs between objective function values in multi-objective optimization in proton therapy. The study seeks to utilize multi-objective optimization to improve the accuracy of intensity-modulate proton therapy, which utilizes pencil beams which have dosage amounts associated with points along the beam (S.-J. Chen and Hwang 1992d). The goal of the work presented by Fonseca is to explain the known issues in evolutionary multi-objective optimization problems and how real world problems involving multi-objective optimization problems are often difficult to solve due to conflicting requirements of the objective function. According to Fonseca, the overall goal of multi-objective optimizations turns into a level of acceptance for the parameters of the objective function that are the result of a compromise in value between the objective function parameters (Fonseca, Fleming, and others 1993). The study of multi-objective optimization is currently assisting scientist in their studies not focused only on manufacturing design, but also in areas such as robot AI, evolutionary life cycles, and proton therapy techniques.

Multi-Objective Optimization With High Fidelity Metamodels

The concept of multi-objective optimization can be applied to objective functions values that are created using high fidelity models, but it is suggested that a designer use metamodels to assist in computation speed. Because Multidisciplinary design optimization (MDO) is a complex method of optimizing the design when creating a design composed of multiple subsystems. When working on optimization, the amount of computational resources increases when seeking a higher level of fidelity in the analysis of subsystems. It is helpful to develop metamodels to use during analysis rather than using actual solvers to reduce the amount of computation required to analyze multiple subsystems. In an example study using metamodels for high fidelity objective function calculations, J. He used metamodels during the optimization of a ship hull. J. He used metamodels instead of actual solvers for resistance, seakeeping, and maneuvering. By using a metamodel, HE was able take the time required to perform optimization on the design of a ship hull from hours to seconds for all points produced by the model. Then HE used actual solvers on what was believed to be the optimal solution in order to verify that the solution produced by the metamodel was correct. The use of metamodels allowed HE to analyze a far greater number of solutions in a much shorter time frame as there was no longer a need to run the software for 12 hours for each solution (He, Hannapel, and Vlahopoulos 2011). By using meta-models, a designer can apply the concept of multiobjective optimization to objective function values that were created using high fidelity models.

Methods for Searching the Solution Space with Multi-Objective Optimization

In multi-objective optimization, there are many methods for identifying robust points in a set of solutions by generating random points around a solution. This point generation can be performed using Monte Carlo methods, which is potentially inefficient as the same solution can potentially be used twice, or by using the concept of a Latin hypercube. Deb uses the Latin hypercube concept to generate patterns of points around a solution to identify robust points in an example of his first method at the start of each generation of the first method a new random Latin hypercube of points is generated around each point and a random point is chosen to calculate the mean effective objective function value for each group of points (Kalyanmoy Deb and Goel 2001b).

Deb isn’t the only researcher looking for ways to more efficiently search the solution space. Josephson presents a method that utilizes three serial modules to explore a large solution space. The names of these three modules are seeker, filter, and viewer. The seeker module is used for selecting from the list of available components. The seeker module also ensures that the configuration of these components satisfies given constraints placed the design. This method of design space reduction ensures that computational resources are not spent on further evaluation of designs that would not produce a viable or would produce a sub-optimal result. Once designs are configured then the filter module uses a dominance based preto-optimal reduction method is used to reduce the number of designs. Dominance filtering is filtering out designs that are dominated by at least one other design. Dominance filtering reduction method produces a best-in-class set of designs during each iteration of the design analysis. Last the viewer module is used to provide designers with a visual means of identifying interesting areas of the solution space. The viewer is especially useful for human interaction with multiple criteria parameters in which optimization of the solution space is difficult to procedurally determine (Josephson et al. 1998).

In addition to the seeker, filter, viewer method described above, Josephson also describes a method of design space exploration that takes a broad range of samples from all areas of the design space. The goal is to ensure that all areas of the solution space are sampled well enough as to give a reasonable estimation of the design space. It is possible that sampling from all regions of the design space may result in a very large number of designs string computational resources may be required for solution space exploration (Josephson et al. 1998). When dealing with this kind of design space exploration, a designer is dealing with an embarrassingly parallel problem. This means that parallel processing will be able to search all areas of the design space at once given enough compute power.

Reducing the Problem Space For Multi-Objective Optimization Problems

Emmerich proposed that order to reduce the size of the problem space in multi-objective optimizations problems, the method of using equality and inequality constraints can be utilized (M. T. M. Emmerich, Giannakoglou, and Naujoks 2006). An example of using an equality constraint is listing that $x = 2$. This simply states that in all solutions to the multi-objective optimization problem, the variable x will always be equal to 2. An example of an inequality constraint is listing that $y < 3$. This means that in all solutions to the multi-objective optimizations problem, y will always be less than 3. Putting both the equality constraint and the inequality constraint together would result in a problem space in which x was always equal to 2 and y was always less than or equal to 3. Equality and inequality constraints are simple concepts to follow but the user of these methods must be careful as to not eliminate areas of the design space that could hold the desired result.

The method presented by Kang in "An Approach for Effective Design Space Exploration" is a method for efficient exploration of a design space using a user-defined metric for reducing the number of solutions which require analysis. The reduction of the number of solutions requiring analysis is performed through identifying similar solutions and using analysis of one solution to represent the probable value outcome of analysis on other similar solutions (Kang, Jackson, and Schulte 2010). As Kang describes, an effective design space exploration (DSE) tool must utilize an effective means for representation, analysis, and an effective exploration method. Representation is ensuring that the data is well represented without requiring the analysis of every solution. Proper *Analysis* must be able to ensure solutions are valid and be able to handle potentially complex calculations for determining feasibility and constraints. Last, exploration must be able to effectively eliminate inferior solutions. By utilizing this guideline for a DSE tool a user can explore their data efficiently for useful solutions (Kang, Jackson, and Schulte 2010).

Utilizing a Vector in Multi-Objective Optimization

Studying the use of vectors in multi-objective optimization is proposed by several authors for different applications. Kuroiwa presents a method in which a vector representing the worst case values for each component in a multi-objective optimization problem (Kuroiwa 2001). Fliege presents the same concept but applied to portfolio selection problems(Fliege and Werner 2014). Yu also presents the same concept but applied to game theory (Yu and Liu 2012) . Vectors when searching for solutions in multi-objective optimization problems are applicable to a wide number of applications as seen in the works presented by the authors above.

Uncertainty and Imprecision

Uncertainty and Imprecision play a large role in the manufacturing process. After modifications have been made to the initial design, eventually an acceptable design will be created which can include information on manufacturing imprecision and uncertainty. The issue with having this information on imprecision and uncertainty is that the tools designers use often do not have any way of capturing these imperfections in the manufactured final product (K. Deb et al. 2002). In order to explore the differences in between uncertainty and imprecision, the following section will be divided into two sections. The first section will focus on uncertainty and methods associated with dealing with uncertainty. The second section will focus on imprecision and the ideas for handling imprecision in the manufacturing process.

Uncertainty

Uncertainty is uncontrolled variations in manufacturing. In naval ship design, the first ship may turn out to be 152 feet long and the next ship built using the same specifications may turn out to be 148 feet long. It is common to have minor differences in the end product of any engineering design. It is because of dealing with these differences that the field of uncertainty in engineering design is a common area of research. For example, Chen uses the same concept of representing uncertainty in multi-objective optimization problem as Kuroiwa and applies it to proton therapy for cancer treatment. The method utilizes preto fronts to identify the tradeoff between properly dosing the intended target with radiation versus the potential for harm to unintended targets in proximity of the target (S.-J. Chen and Hwang 1992d).

Doolittle also shows interest in handling uncertainty in a method of replacing objective function values with a value the represents an uncertain multi-objective optimization function value. This method also includes constraints that are placed on the objective function values (Dolan 1989). Gunawan also displayed interest in handling uncertainty in multi-objective optimization problems. The method proposed by Gunawan uses what was called a *sensitivity region*. The sensitivity region is the region that contains possible solutions to the allowed variation of uncertain parameters. The method presented by Gunawan is used to identify preto optimum solutions in a discontinuous and/or non-differentiable front (Gunawan and Azarm 2004). The target area of research for the Gunawan's work is on a vibrating platform. The result of this Gunawan's work shows a method in which a designer can identify points in the decision space that can handle small perturbations to their value by using a sensitivity region around the point. Handling uncertainty in manufacturing is a valued area of research and will help to lessen the potential of design failure after construction.

Imprecision

Imprecision is the unavoidable vagueness in the objective function values in the beginning of the design process and leads to fundamental difficulties in identifying a resilient design in multi-objective optimization. Imprecision is a fundamental problem in multi-objective optimization problems because there are often many options available to a designer, so trying to identify components that will all work together to achieve the goals of the design is a complicated process. In order to deal with imprecision in manufacturing, Antonsson describes the Method of Imprecision or MOI. MOI is a method based on fuzzy mathematics used for handling imprecision in design methods in engineering. MOI is a useful tool in set-based concurrent design and in the preliminary stages of engineering design (Josephson et al. 1998).

Closing of Uncertainty and Imprecision

There is a distinct difference between imprecision and uncertainty in engineering design (K. Deb et al. 2002). Imprecision is having a range of possible values for a particular parameter but having no way of being able to determine the exact value that will be chosen for the parameter in the final product. Imprecision is inherently and unavoidably part of the initial design process. Designers often start with many options for a given component of a design. It is unlikely that the designer will know the exact optional component that will be used in the final product due to the relationship between components in a multi-component design. Imprecision lends to the goal of designers to study multi-objective optimization by giving a wide range of options for components in a design.

Compensating Methods

Compensating methods are utilized in multi-objective optimization problems in order to alleviate the strain of some parameters not performing as well as others. Compensating methods often use combination functions in order to identify designs that may not be as strong in one parameter but good enough in another parameter to make up for the weaker parameter. To give a little more detail on combination functions, combination functions are functions that combine objective values of the objective value to give a combined score to the objective function. The score is a combined means for describing how well an objective function will perform on a given task. The name of the function that combines the objective values into an aggregate score is often called a *metric*.

Combination functions can be observed as two types and those two types are compensating and non-compensating (Josephson et al. 1998). In compensating combination functions, the function will compensate for objective function values that do not perform well with objective function values that perform very well. For a non-compensating combination function, the objective function will not compensate for attributes that perform poorly and as a result will have a objective function value that is limited by the worst performing objective value.

Minimizing the effect of the weakest parameter is a goal of compensating method user. The adaptive weighted sum method or WSM was created to detect uniformly-spaced Pareto optimal solutions. The adaptive weighted sum method was designed to provide an adaption to the commonly known weighted sum method, which is the most commonly used algorithm in multi-objective optimization problems (Kim and Weck 2006). The weighted sum method is performed by multiplying all objective functions by a weighting factor and adding up the weighted objective functions. The weighted sum method has some pitfalls and one of the pitfalls is the inability to handle non-convex portions of a Pereto surface. One of the features of the adaptive weighted sum method is that it can reach points in the non-convex portions of a pareto surface. In addition, the weighted sum method ignores the non-Pereto Optimal solutions and it can handle problems with two or more objective functions. The adaptive weighted sum method works by using a two-phase process. In the first phase the algorithm uses the weighted sum method to identify Pereto front patches and in the second phase, additional constraints are placed on the Pereto front patches. The additional constraints are used to refine the patches in order to create a well distributed preto front mesh.

Weighted Sum method: $O_{\text{weighted Sum}} = W1O1+W2O2+W3O3+W4O4 + ... + WxOx$

Utility theory is an addition to compensating method, which is a method of creating a weighted sum that includes uncertainty. To briefly review, a weighted sum is creating an aggregate of objective function values that are used to give a score to the objective function. Utility theory adds a little too weighted sum method as it also includes uncertainty. Because Utility theory is an aggregate of the objective function, it can view viewed as a compensating method for assigning objective function value. The reason why Utility theory is considered a compensating method is because an objective function can have a low or zero score for one of its objective function values and still register an acceptable objective function score by scoring high for another objective function value.

Methods have been created in order to assist Utility theory in avoiding the issue of having a objective function be considered acceptable even though it contains an objective function value that scores below what would be acceptable for the individual objective function value. Two of these methods are objective constraints and subjective goals. When using objective constraints, the values of the objective function must meet specific guidelines without relying on any of the other objective function values. For subjective constraints, objective function values are able to trade values between other objective functions values in order to meet the overall requirements of the tradable objective function values or for the overall objective function.

Pareto Front

The concept of the Pereto front was developed by Vilfred Pareto (1848-1923) and can be read about in the 'Manual of Political Economy' [TODO: Cite]. The idea of the perato front in relation to multi-objective optimization is that a point on the pareto front cannot increase the value of any objective function without decreasing the value of another objective function value. Pereto fronts have been heavily studied and utilized for exploring solution spaces. We now present several methods which applied the usage of Pereto fronts.

Using the idea of a Pereto front has a pitfall of it being possible to lose the perato front optimal solution during optimization, Goel proposes a method to quantify trade-offs among objectives in the comprised region. He proposes a "methodology to construct a response surface approximation of the Pareto optimal front based on surrogate models." Geol explains that during optimization of an elitist non-dominated Multi-objective evolutionary algorithm (MOEA), it is common for the population size to exceed the size of the original population. When the population size exceeds the original population size the non-dominated solution are lost and during this loss, it is possible that the preto optimum solution can be lost without hope of recovery of the lost solution during optimization (Goel et al. 2007). This loss is known as Preto-Drift. Deb presents an algorithm to assist with Preto-Drift. The algorithm is called NSGA-II and this algorithm works by storing all non-dominated solutions of optimal preto fronts in an archive format in order to improve convergence of the preto optimum front. By storing all non-dominated solutions, the time and memory required to compute the preto front is increased but the Preto-Drift is reduced (Daum, Deb, and Branke 2007).

Deb discusses an interesting aspect of Pareto front multi-objective optimization problems in his paper “A Hybrid Integrated Multi-Objective Optimization Procedure for Estimating Nadir Point.” The nadir point is a point representation of the objective function with the lowest possible objective function values corresponding to the Pareto front (Kalyanmoy Deb and Gupta 2006). As Deb points out, the Nadir point is often incorrectly describe as the combination of the lowest objective values for all points in the design space which results in an overestimation of the Nadir point (Kalyanmoy Deb and Gupta 2006). The Nadir point is a significant point because it is used to identify the range of possible values for the objective function. The range of acceptable values from the Pareto optimal front to the Nadir point can be visualized using methods such as bar charts, petal diagrams and value (Kalyanmoy Deb and Gupta 2006). Once a designer has the range of acceptable values using the Nadir point to the Pareto front points, the designer has the option to normalize the points using a method described in *Nonlinear multi Objective optimization by K. Miettinen.* The Nadir point is also a rather difficult point to locate in objective functions with 3 or more objective function values as it often requires a clear understanding of the design space which is not always easy to obtain due to the inherent imprecision associated with early stages of multi-objective design optimization (Miettinen 1998).

Component-based design is a method of design in which standard components are assembled to completed a design. Component-based design can be thought of as building a design using a predefined set of building blocks. Computer assistance is especially useful when using component-based design.

Using the idea of maximization and minimization tradeoffs as understood from the Markowitz portfolio optimization problems, Fliege presents a method for locating the robust preto front in a multi-objective optimization with uncertainty problem. The method utilizes standard methods in multi-objective optimization to locate the robust preto front (Fliege and Werner 2014).

Robust Solution

Choosing points which lie in the Pereto front of a solution space can come at a price. The points which lie in the Pereto front are often not resilient to change. This means that any modification to the values of the objectives in the objective function will result in a point that is no longer feasible. A feasible design is a design that will accomplish the overall objective of the design. While the points lying on the Pereto front may be the best for short term multi-objective problems, there is a danger in choosing Pereto front points in objective optimization problems in which changes can happen to the objective function values after the Pereto front has been identified.

In multi-objective optimization problems, a large number of algorithms are focused on finding the global optimum solution or the preto front of optimal solutions, however, in practice, it has been found that the optimal solution is often sensitive to perturbation in its value. In practice, designers are often more interested in points that can withstand small perturbations to its value and therefore produce a stronger solution. The way that Deb intends on finding robust solutions is to take the mean value of a solution based on points within its vicinity. This will result in a point that is more robust because it is comprised of several points (Kalyanmoy Deb and Gupta 2006).

Deb presents two methods in the described work. The first method seeks to use the normalized difference between the function value and the perturbed value. If the normalized difference is less than the chosen thresh hold then the function is found to be robust. The second method seeks to use the mean effective function value or a value representing the level of acceptable perturbation in the objective function values (Kalyanmoy Deb and Gupta 2006). Deb's method gives the user control over acceptable robustness level of function solution.

According to Gunawan, there are two major types of optimization approaches found in literature. The two type are deterministic approaches and probabilistic approaches. "Deterministic approaches obtain a robust optimum design using its first-order derivative or other non-statistical measures, and then incorporate such measures when optimizing the design objective (Gunawan and Azarm 2004)." Probabilistic approaches use statistics to gauge the level of sensitivity (commonly used method are mean and variance) of a design and then use the results of these statistics to "optimize the design based on this information (Gunawan and Azarm 2004)."

In "Introducing Uncertainty in Multidiscipline Ship Design" Hannapel discussed the importance to identifying constraints influenced by uncertainty during the optimization process. Once the constraints are identified, the concept of reliability can be applied thus converting the uncertainty constraints into probabilistic constraints. The end result is of the process of introducing reliability helps ensure that the determined solution will provide a probabilistic result within a given reliability level. Robustness is introduced into the optimization process by "modifying the objective function to depend on the mean and variance of the response of the objective function" (Hannapel and

Vlahopoulos 2010). The focus of the paper was to introduce reliability and robustness into a multi-disciplined parallel optimization process containing properties with uncertainty in the ship design process.

Ehrgott, Deb, Gunawen, and Hannapel all discuss methods for robust optimization. Finding the robust solution is a leading reason for study in the area of multi-objective optimization problems. A robust solution is a solution that is capable of withstanding changes to parameter values. Robust solutions are often desired over optimal solutions due to the inability of most optimal solutions to withstand perturbations to solution parameter values. We now present multiple methods used in searching for robust solutions.

Scoring Distance

The work by Barrico presents a method of using the distance between points in a solution space to determine the degree of robustness. Distance between points could be used to determine areas of point concentration and could mean that points located in these areas had feasible possible values within range (Kalyanmoy Deb, Miettinen, and Sharma 2009).

Deb applies the concept of applying difference of mean value and original objective function value to multi-objective optimization based on original method proposed by Branke for single optimization Barrico adds to Deb's method by adding degree of robustness which is based on neighborhood of objective function values (Kalyanmoy Deb and Goel 2001a). Branke proposes a method for single objective function that assigns a mean value to each objective function value based on a predetermined neighborhood of values (Bernstein 1998). Deb utilize the method proposed by Branke but instead of applying it to single optimization functions, they apply the Branke method to multi-objective functions and they also present an idea for restricting the difference between the mean value of the objective function and the value of the original objective function. The result of the first and second multi-objective optimization concepts by Deb is the ability of the designer to predetermine the level of robustness they would like to achieve (Correa Florez, Bolaños Ocampo, and Escobar Zuluaga 2014).

Taking the first half of the method presented by Deb for applying a mean value to each objective function value in a multi-objective function, Barrico proposed the *degree of robustness* to extend Deb's method. The *degree of robustness* is performed by take a neighborhood of objective function values and applying a ratio to each objective mean value and to not allow objective function mean values that lie outside the range of the ratio (Kalyanmoy Deb, Miettinen, and Sharma 2009). The goal of this work by Barrico is to locate the non-dominated front of robust solutions in a trade space. The work utilizes the methods of neighborhoods to calculate the robustness of a point. Neighborhoods of increasing distance are calculated around a point. The number of times the distance of the neighborhood around the point is increased is part of the degree of robustness calculation. While the robustness level of a point is less than a given threshold, the size of the distance will be increased until the threshold of the level of resistance is met (Kalyanmoy Deb, Miettinen, and Sharma 2009). Using a distance calculation which look at the distance from every point to its neighboring solutions is an interesting concept and this work provides a useful and easy to follow method of utilizing point distance for calculating robustness of a point.

Scoring Distance into Genetic Algorithm

In Deb's method for finding a robust preto front, for each point in the objective space, random points are generated within a set vicinity to determine a mean value for each point. This mean value represents how well a point can withstand changes to its value within the given range. Each point in the objective space becomes a representation of the mean value of points within a given range and after a few thousand generations of NSGA-II, a good understanding each point's ability to withstand minor changes is obtained (Daum, Deb, and Branke 2007).

This method by Deb could be used in addition to the substitution method that I have proposed in order to add a stronger sense to the idea of robust optimization. After the substitution method with other viable objective function values, points could be generated randomly within a range of each solution and the local mean value could be assigned to each point. Not only would a solution show whether it could withstand changes by having other viable objective function values substituted for its own, but a solution would show its resilience to minor modifications to its objective function value within its local given range.

Avigad searches for a solution to unconstrained multi-objective optimization problems using an evolutionary algorithm. Avigad discusses a method in which solutions are associated with a performance cluster. This cluster represents how a solution may perform in relation to a set of solutions with similar characteristics. In order to find the "best of the worst case" set of performance clusters, Avigad uses an evolutionary multi-objective optimization algorithm (EMO). EMO algorithms have been found to be very useful in locating solution space fronts in large data sets so this is why Avigad chose this method (Farina, Bramanti, and Barba 2002)

By providing an answer to a solution based on the set of worst case solutions, the user has an idea of how solutions within the set will perform. Using a set of worst case solutions is also preferred over trying to select a single worst case solution because different designers will have different ideas on which component is the most valuable. In addition, Avigad also presented work that was focused on determining the amount of distance a solution needs to be shifted in order to be no longer dominated by another solution (Farina, Bramanti, and Barba 2002).

In addition to the Worst-Crowded NSGA-II method, Deb also describes the Extremized-Crowded NSGA-II Approach. This method, like the Worst-Crowded NSGA-II Approach, uses sorting to assign rank but it assigns rank values in a slightly different manner. The sorting is performed on the population and the rank is assigned based on distance from the closest extreme point. The WC NSGA-II Approach uses the members in each generation of the population on every non-dominated front. It takes these population members and sorts them from minimum to maximum based on each objective function value. The WC NSGA-II Approach then assigns a rank to each objective function member based on its rank in the list. In this work using the substitution method, we also sort the list of the population objective function values based on the resilient score of each member of the population and then we assign each member a rank. As Deb explains in the WC NSGA-II Approach, assigning rank to the members in the population after sorting based on objective function value, ensures that the maximum objective value will receive the best crowding distance score (Daum, Deb, and Branke 2007). Using this idea of sorting to assign rank for crowding distance score can be directly associated with the idea we used in the substitution method for sorting by resilience score and assigning rank.

Genetic algorithms are useful for exploring solution spaces in which the solution space is non-linear, discontinuous, non-differentiable. This solutions provided genetic search algorithms do not guarantee an optimal solution but they will provide a solution the is considered to be near-optimal. A common technique in genetic algorithms is known as crossover. This method seeks to acquire the best attributes from both parents to produce a stronger child.

The payoff table is a method in which the objective function values are plotted into a table format making it easier to view the relationship between the objective function values (Kalyanmoy Deb, Miettinen, and Sharma 2009). The payoff table suffers from the limitations of having the possibility of an incorrect Nadir point determined by identifying inaccurate minimization of the objective function values. An example of inaccurately identifying the Nadir point region can be seen in figure 2 below by looking at the dark shaded section of the solution space.

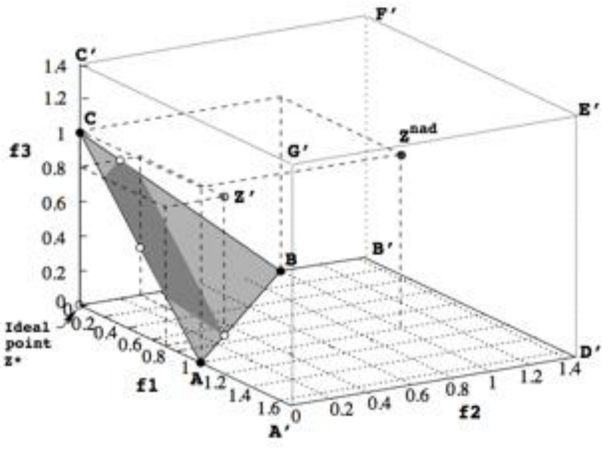


Figure 2. Payoff table by Deb

A payoff table may not produce the true nadir (Kalyanmoy Deb, Miettinen, and Sharma 2009)

Deb's points out that because it is possible that the payoff table method can locate and inaccurate estimation of the Nadir point that a more reliable method is required. Several methods for better estimating the Nadir point are presented by Deb (Kalyanmoy Deb, Miettinen, and Sharma 2009). The methods presented were the Worst-Crowded NSGA-II method and the Extremized-Crowded NSGA-II approach. The comparison of these two approaches resulted in showing that the extremized NSGA-II approach was able to reliably calculate the nadir point for multi-objective optimization problems up to 20 parameters.

Set Based Design

In the paper 'What is Set-Based Design', the Singer's goal was to describe set-based design and how it relates to naval ship design. Naval ship design has traditionally been done using the point-based design method. The author explains the set-based design method and how it improves over the point-based design method (David J. Singer, Doerry, and Buckley 2009).

In long term problems of objective optimization, such as naval ship design, changes to the values of the objectives that make up an objective function is all but guaranteed. For example, throughout the life cycle of a naval vessel, that vessel will always become heavier. This is due to components being added to the naval vessel and modifications to the initial configuration of the naval vessel. As the naval vessel becomes heavier, that vessel is no longer able to move at the same max speed that it was able to achieve early in its life cycle. This is due to there being more weight for the engines of the naval vessel to have to move. In problems of multi-objective optimization in which changes can happen to objective values, a designer should be aware of the impact of those changes to the objective values in the overall design.

Description of Set-Based Design

Traditionally the process of designing complex systems happened in what is known as the point-based design method. In each step of the design process, an choice would be made based only on whether that element fit within the constraints placed on it from previous elements in the design process (Figure 3). For example, if a designer's chose this weapon system now then the designer can only choose radar system A or B in the next step of the design process. The point-based method worked and was successful but the method possesses some pitfalls. The pitfalls are that as designers are choosing elements that fit into their design that it is possible to fall into a situation where the only choice for the current step in the design process will invalidate a previous. When a previous choice becomes unfeasible it causes the designers to start back at the point of the design process with the new invalidated previously valid choice and make new choices until all choices in the design sequence are valid. This re-choosing of points could cost weeks to months of development time. Eventually, the designers would find a design with all feasible choices made at every step of the point-based design method, but the final product of the design was most likely not an optimal design. The design chosen using the point-based method was most likely only a possible valid design.

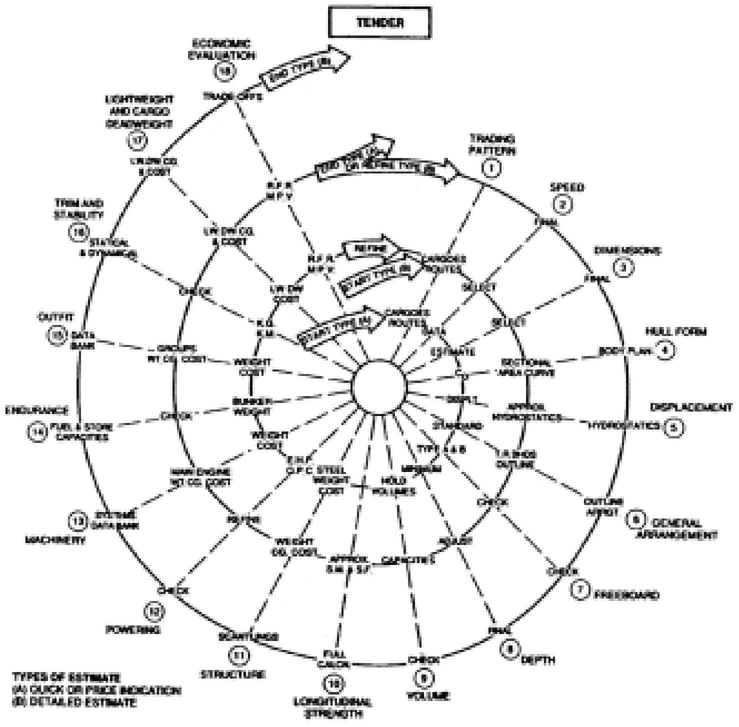


Figure 3. Classical Design Spiral by Evans

A choice would be made based only on whether that element fit within the constraints placed on it from previous elements in the design process (Evans 1959)

Point-based strategies consist of five basic steps (J. K. Liker et al. 1996):

1. First, the problem is defined.
2. Engineers generate a large number of alternative design concepts, usually through individual or group brainstorming sessions.

3. Engineers conduct preliminary analyses on the alternatives, leading to the selection of a single concept for further development
4. The selected concept is further analyzed and modified until all of the product’s goals and requirements are met
5. If the selected concept fails to meet the stated goals, the process begins again, either from step 1 or 2, until a solution is found

Set-Based Design (SBD) method is an improvement over the point-based design method. SBD ensures that the design chosen after all choices of variables in the design have been chosen from their set possible values in the optimal range. The reason why the points chosen from the SBD method are able to be chosen from their optimal value ranges is because more time is spent analyzing the range of possible values for a particular variable. Also, the value ranges for each of the variables is studied independently of all other systems in the design. This allows for multiple groups of designers who specialize in different aspects of the design to work independently from one another thus making the problem easier as optimal ranges for all variables can be found without worrying about whether a particular design is incompatible with previous or future components in the design. A good example of the process of SBD can be seen in Figure 4 that shows independent groups of design specialist starting off their design in separate areas and then combining their efforts to produce a single more optimal design. Because designers have values in optimal ranges for all variables in a system, it allows designers to chose an optimal design.

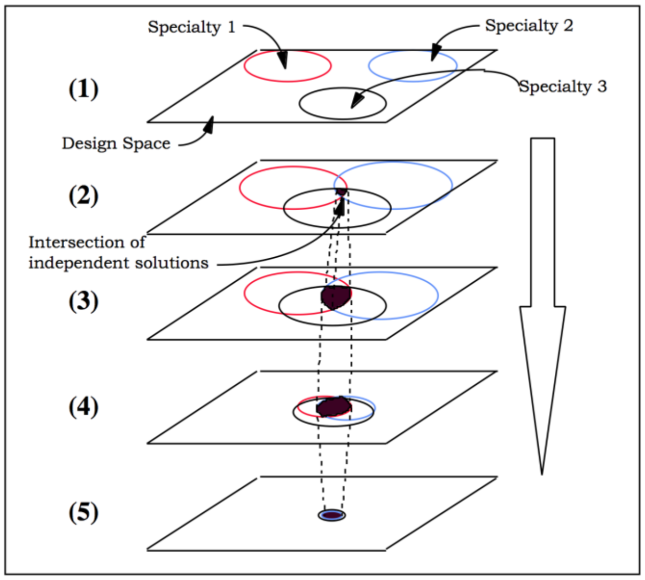


Figure 4. Set-Based Design Process

(Bernstein 1998).

Set-Based Design fits into the area of Concurrent Engineering “is one step beyond Point-Based Design (D. J. Singer and Parsons 2003). Concurrent Engineering is a method of design in which a team composed of a multiple specialist and different areas of expertise are combined into a single group to develop a better design.

Set-Based design main features include (D. J. Singer and Parsons 2003)**:**

- Broad sets of design parameters are defined to allow concurrent design to begin
- These sets are kept open longer than typical to more fully define trade-off information,
- The sets are gradually narrowed until a more globally optimum solution is revealed and re- fined
- As the sets narrow, the level of detail (or design fidelity) increases.

Because set-based design operates in a manner that is not common to traditional design processes there “has been a source of confusion” as to how SBD is useful (D. J. Singer and Parsons 2003). The confusion comes from the delay in making critical design decisions. By delaying design decisions until a better understanding of the possible solution space for all components in a design allows the designers to make better choices for the final design. This delay in design choices until a better understanding of the solution space is understood is described in ‘The Second Toyota paradox: How delaying Decisions Can Make Better Cars Faster” (Technology and reserved 2016a). This paper describes how Toyota is able to design cars using SBD methods faster, more efficient, and creating a better product as the final design than if they had used traditional point-based design methods like their competition.

By allowing more time for the designers to make critical design decisions in their area of expertise, the cost associated with the design process are kept much lower throughout the design process. An example of the lower cost throughout the design process can be seen in figure 5**.** By taking time more time to develop their design and making more optimal choices at every step in the design process, designers are able to be more efficient in their design choices. Being more efficient in the design choices prevents the need to remake components that are no longer viable in the current iteration in the design from going into production. Efficient design choices generate a lower overall cost in the labor associated with developing a design at both the production levels and the design levels.

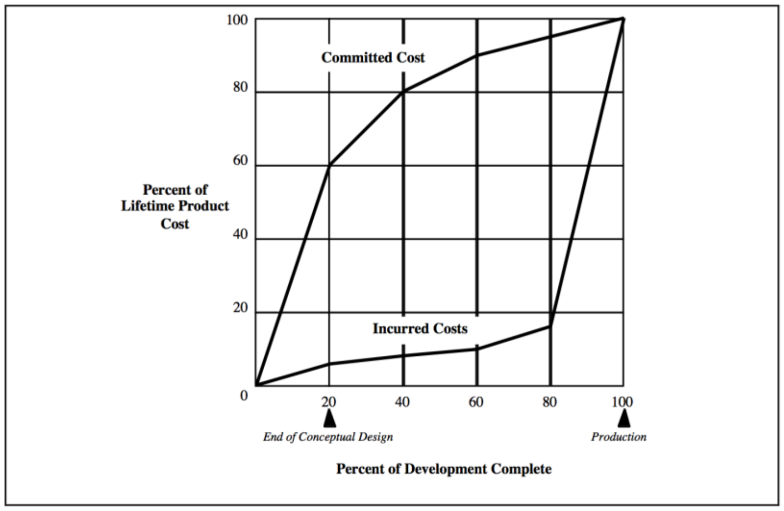


Figure 5. Designing-In Costs

(Bernstein 1998)

During the initial stages in the design process, the stakeholders have a critical level of impact on the final design. Often times the stakeholders will make choices on critical components of a design when there is little data on the impact of those design decisions. An example of the amount of knowledge through the design process can be seen in figure 6. These stakeholder choices at the early stages of development with little knowledge of the impact of those choices can have lasting impact on the overall design.

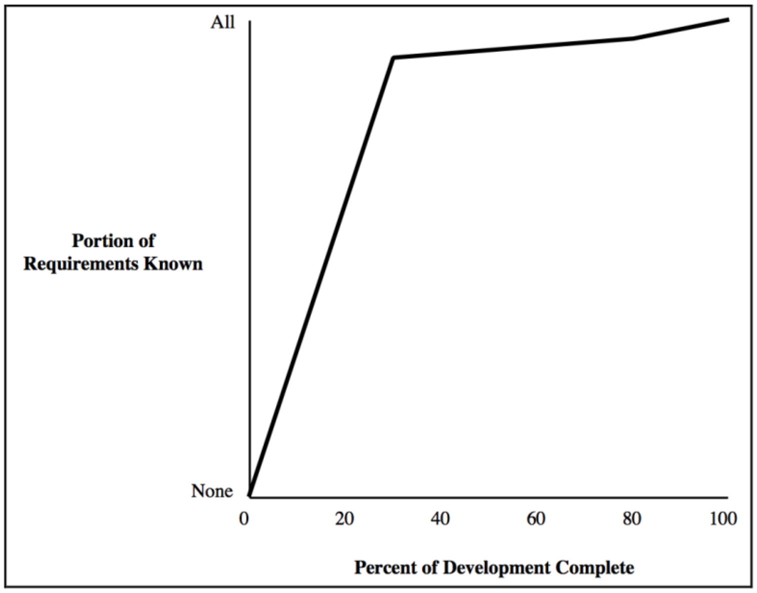


Figure 6. Evolution of Design Knowledge

(Bernstein 1998)

The set-based design method is a method in which multiple designers work on their specialized area of the design without worrying about how their design affects other parts of the overall design. This allows specialized designers to focus on their area of expertise by allowing the designers to create analyze the set of best possible options in their area of the design. Since designers at all stages are able to identify the best options for their area of expertise, the overall design of the ship is improved.

The point based design method is the process of choosing components in a sequential order without cause the entire system to become unfeasible. An example of an infeasible design would be a ship that no longer floats. When designers are forced to or opt to choose a component that causes the ship to become unfeasible, the designers must return to a previous component and choose new components until all components for the overall ship design are chosen and the result is a feasible ship.

Point based design is the method of trying to find a single solution that meets all requirements of a design. Design decisions are made in sequence and often require backtracking to previous decisions as new requirements of the design become known. Point based design has a key drawback and that drawback is that a feasible design may be located but that design is unlikely to be a global optimum in the design space.

By allowing each design team to focus on areas of the design without having to worry about how their design area affects other steps in the design process, design teams are able produce a more robust design. All design teams for a particular design are able to work on their area of the design concurrently. The result of this design method is that more time is spent searching the solution space of possible options for each area of the design, but a good option for each component in the design is available. Traditional point based design method is a contrast to spending more time in each design phase because point based design makes a less informed design choice at each step in the design process. Point based design is burdened with the issue of having to go through the steps of backtracking through the design process while set based design does not suffer from backtracking through the design process because it has many options pre-prepared for each step in the design process (Sobek, Ward, and Liker 1999).

Set-based design is a method of analyzing a design space by analyzing a set of designs rather than the single point design method used in point-based design. Set-based design allows for greater flexibility and helps with the optimization process by reducing the problem size to a more manageable state. After the problem size has been reduced, point-based design can then be used efficiently for analysis of the remaining problem space (Hannapel and Vlahopoulos 2010).

Toyota’s design method is considered to be a more concurrent engineering method than the design method used by both Japanese and US auto manufacturers. This concurrent approach design is performed without requiring design teams to be collocated which is often considered a requirement by other auto manufacturers. Because design decisions are made by design teams using the whole solution space of designs rather than a specific design provided to them by another design team earlier in the design process, design decision makers are able to choose a design from the set of possible designs which results in an overall better design decision (Morgan and Liker 2006).

The process of using set based design may be difficult for companies to develop. Toyota has developed a long-standing relationship with manufacturers that is built on trust and the knowledge that the manufacturers know specific ranges of values that the components they develop can utilize. Design decisions on how to identify sets of designs are made by senior engineers with 15 to 20 years of experience. These decisions on how to shape the design set is based on years of hands-on involvement in the design process and thus companies that wish to adopt set based design have many years of design experience before implementing set based design (Morgan and Liker 2006).

‘‘The second Toyota paradox: how delaying decisions can make better cars faster,’’ Toyota’s design process is highly effective but seems as though this method would slow down the overall design process as design decisions are delayed until very late into the design process. The traditional method of design is to make design decisions early in the design process and then to refine those design decisions as the design process moves forward. This method of design is known as the point-based design method (Technology and reserved 2016a). Toyota does not use the point-based design method but instead uses the set-based design method and part of the requirement of using the set-based design method is to delay design decisions until a large amount of information is gathered for each component in a design.

Toyota consistently shows a high profit per vehicle and growth in market share. The tools Toyota uses for its development are nothing special but rather the high success rate is due to their design process. Toyota uses what is known as set based design for their design process. This design method focuses on analyzing a large set of designs rather than starting from a specific design and refining that design. Starting from a specific design and refining that design is the most widely used method of design and is known as point-based design. Point based design has many pitfalls such having to revisit steps in the design process many times due to changes in requirements for steps further in the design process. Set-based design avoids most of the headache with design changes by providing many options for each component of the design. Having many options for components allows changes further in the design process by having alternative components ready to go for each step in the design process (Sobek, Ward, and Liker 1999).

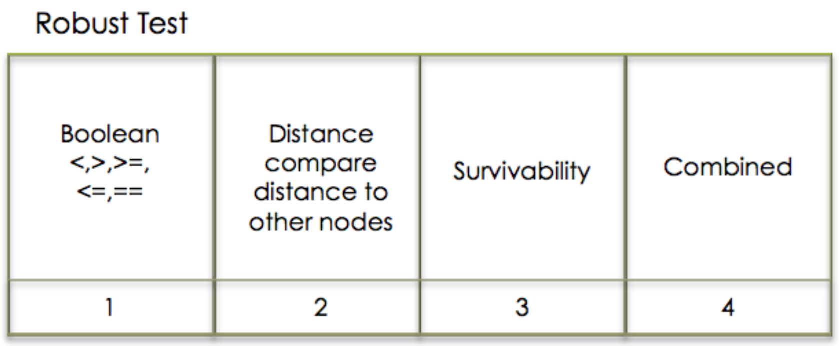
Toyota's design method is considered to be a more concurrent engineering method than the design method used by both Japanese and US auto manufacturers. This concurrent approach design is performed without requiring design teams to be collocated which is often considered a requirement by other auto manufacturers. Because design decisions are made by design teams using the whole solution space of designs rather than a specific design provided to them by another design team earlier in the design process, design decision makers are able to choose a design from the set of possible designs which results in an overall better design decision (Sobek, Ward, and Liker 1999).

Marine design is moving to set based in "A hybrid agent approach for set-based conceptual ship design." Marine design in the US is focused around cross-functional teams using concurrent engineering approaches. As with most traditional design approaches, this method of concurrent engineering was still based around point based design methods. After researching the set based design utilized by Toyota, advanced marine design has begun to also use set based design method in order to make more informed decisions during the design process. The goal of utilizing set based design methods is to "provide a greater probability of achieving a global optimum of achieving a global optimum for the overall design" (Parsons, Singer, and Sauter 2016).

The Navy is using set based design as naval ship design is an evolving landscape in which the design specifications for a particular ship can change at any point in the design process. The point-based design method does not adapt to these changes easily and leads to slowdowns in the design process. Set-based design is a more agile approach and can adapt to an evolving design requirements (Hannapel and Vlahopoulos 2010).

In 2014 the Small Surface Combatant Task Force was formed to study the Modifications to the Littoral Combat Ship (LCS) and to study new design concepts. The paper *Concept Exploration Methods for the Small Surface Combatant* describes the results of that study (Garner et al. 2015). The goal of the study was to analyze the results of modifying the current LCS ship, using the current design, and to examine completely new ship designs. In each of the designs examined, the designers were to examine weapon systems, cost, sensors, and the lethality of "the lethality of the ship to air, surface, and undersea threats" (Garner et al. 2015). The goal was to find a ship design that would meet current mission goals while providing more capabilities than the current LCS design. The study utilized Set Based Design methods in order to create a better design than had previously been possible without using Set Based Design. The resulting design was generated using multiple groups of specialist all working in their area of specialty and after each group of specialist finished analyzing their area of the design, the "configuration Capability Calculator intersected the feasible solutions by the Feasibility Element algorithms" (Garner et al. 2015).

Three Methods for Testing Robustness


Robust Test

Boolean <,>,>=, <=,==	Distance compare distance to other nodes	Survivability	Combined
1	2	3	4

Figure 7. Robust Test

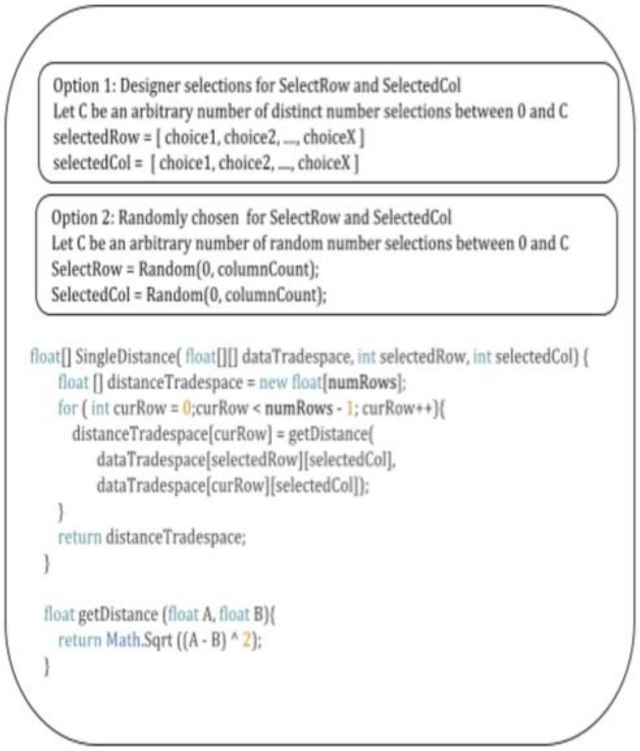
Distance Test Metric for Robustness

The Distance test metric for robustness is performed by examining the Euclidean distance in between points. The Euclidean distance in between points can provide insight into how closely points are related. The idea of measuring the distance in between points is not a new concept as there are many algorithms that measure the distance in between points, however, the focus of this work is to locate a robust design. In order to use the Euclidean distance in between points to locate a robust design, understanding of how methods to calculate the distance in between points using one or more column values, methods for storing those distance calculations, and to calculate a score metric need to be developed.

Three Methods for Distance Calculation

Three methods for distance calculation includes but are not limited to single column distance, multi-column distance, and total distance.

1. Single column distance



```
Option 1: Designer selections for SelectRow and SelectedCol
Let C be an arbitrary number of distinct number selections between 0 and C
selectedRow = [ choice1, choice2, ...., choiceX ]
selectedCol =  [ choice1, choice2, ...., choiceX ]
```

```
Option 2: Randomly chosen  for SelectRow and SelectedCol
Let C be an arbitrary number of random number selections between 0 and C
SelectRow = Random(0, columnCount);
SelectedCol = Random(0, columnCount);
```

```
float[] SingleDistance( float[][] dataTradespace, int selectedRow, int selectedCol) {
    float [] distanceTradespace = new float[numRows];
    for ( int curRow = 0;curRow < numRows - 1; curRow++){
      distanceTradespace[curRow] = getDistance(
          dataTradespace[selectedRow][selectedCol],
          dataTradespace[curRow][selectedCol]);
    }
    return distanceTradespace;
  }

  float getDistance (float A, float B){
    return Math.Sqrt ((A - B) ^ 2);
  }
```

Figure 8. Single Column Distance

Single column distance is calculated between column values in a selected column with each row of data within a tradespace

Storage Methods for Single Column Distance. Understanding methods for storing values during single column distance can be helpful. Next we will discuss two methods for this type of storage.

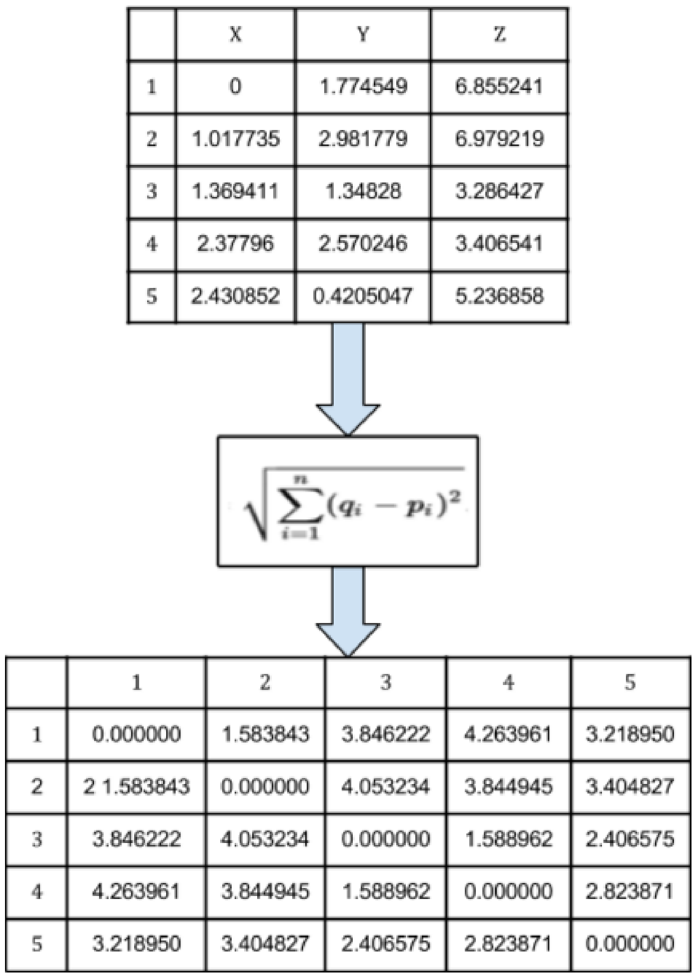
Single Column Distance Selected Value Storage

Single column distance delected value storage is performed with an additional column that holds the distance between a selected column value or a randomly selected column value and all other column values.

Single Column Distance Total Value Storage

Single column distance total value storage is handled by additional column to the tradespace. Each row of data in the additional column holds a matrix. Each matrix stores the result of measuring the distance between each column value in the selected column. Since calculating the distance between all rows of data and all other rows of data in the selected column could be computationally expensive, it is acceptable to choose manually or randomly a set of row values from the selected column to use for calculating distance. However, it is important to remember that not calculating the distance between all points will result in an estimate for the distance calculation. An additional column should be added to the tradespace with the value showing the percent of values that have had distance calculations performed.

Multi-column distance. Multi-column distance is the distance between values in two or more columns values in each row of data in the tradespace.



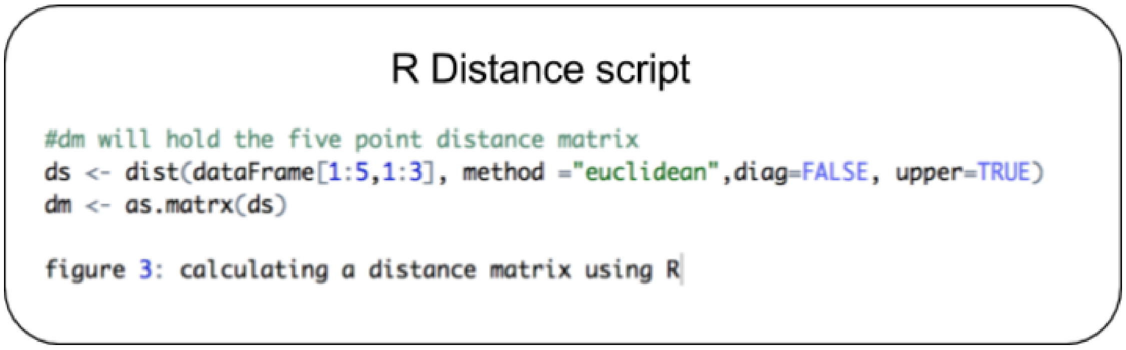
	X	Y	Z
1	0	1.774549	6.855241
2	1.017735	2.981779	6.979219
3	1.369411	1.34828	3.286427
4	2.37796	2.570246	3.406541
5	2.430852	0.4205047	5.236858

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

	1	2	3	4	5
1	0.000000	1.583843	3.846222	4.263961	3.218950
2	2 1.583843	0.000000	4.053234	3.844945	3.404827
3	3.846222	4.053234	0.000000	1.588962	2.406575
4	4.263961	3.844945	1.588962	0.000000	2.823871
5	3.218950	3.404827	2.406575	2.823871	0.000000

Figure 9. 3D point data converted to Euclidean

At the time of the writing of this work, the statistical language R provides an easy to use function that will handle the creation of this distance matrix.


R Distance script

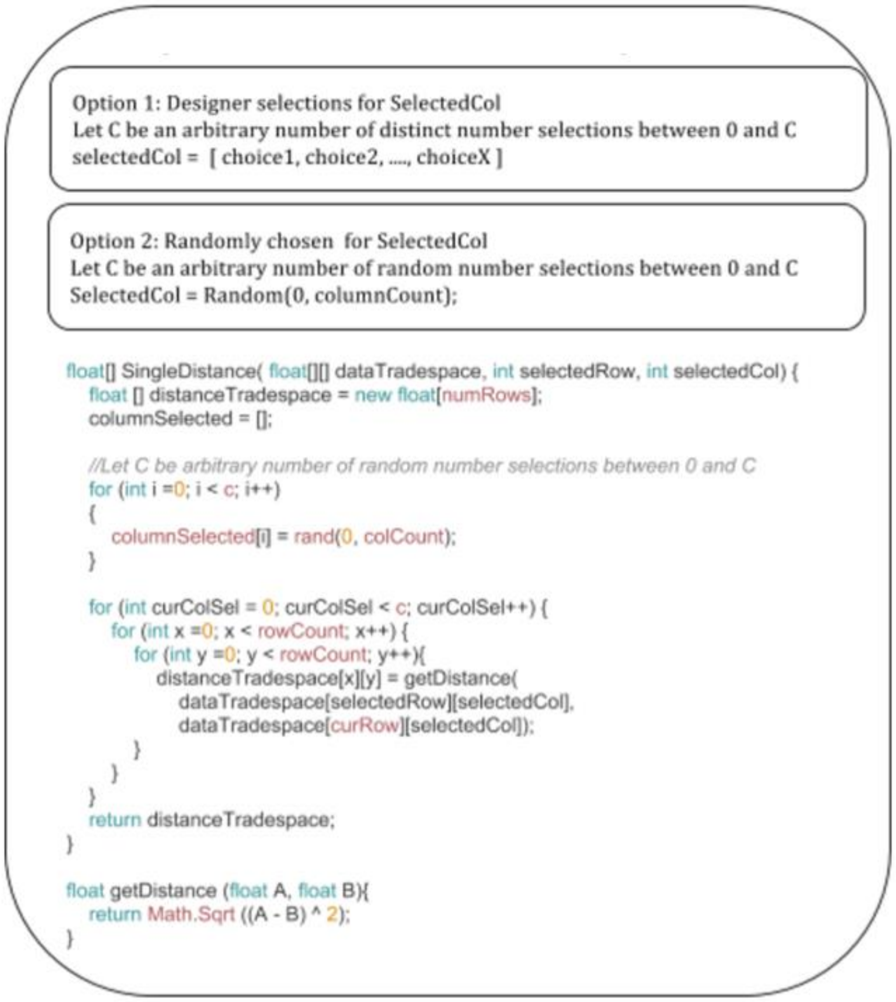
```
#dm will hold the five point distance matrix
ds <- dist(dataFrame[1:5,1:3], method ="euclidean",diag=FALSE, upper=TRUE)
dm <- as.matrx(ds)

figure 3: calculating a distance matrix using R
```

Figure 10. R distance Script

Because the designer is choosing multiple columns to work with in Multi-column distance, the designer has options on the storage of the value of the distance calculation, and on the metric used for calculating the score value for the function. The following are two potential storage options for the values resulting from the distance comparisons.

Individual Distance Storage

Individual distance storage is a method of storing the individual distance value for each comparison of column values using two or more columns.



Option 1: Designer selections for SelectedCol
Let C be an arbitrary number of distinct number selections between 0 and C
selectedCol = [choice1, choice2, ..., choiceX]

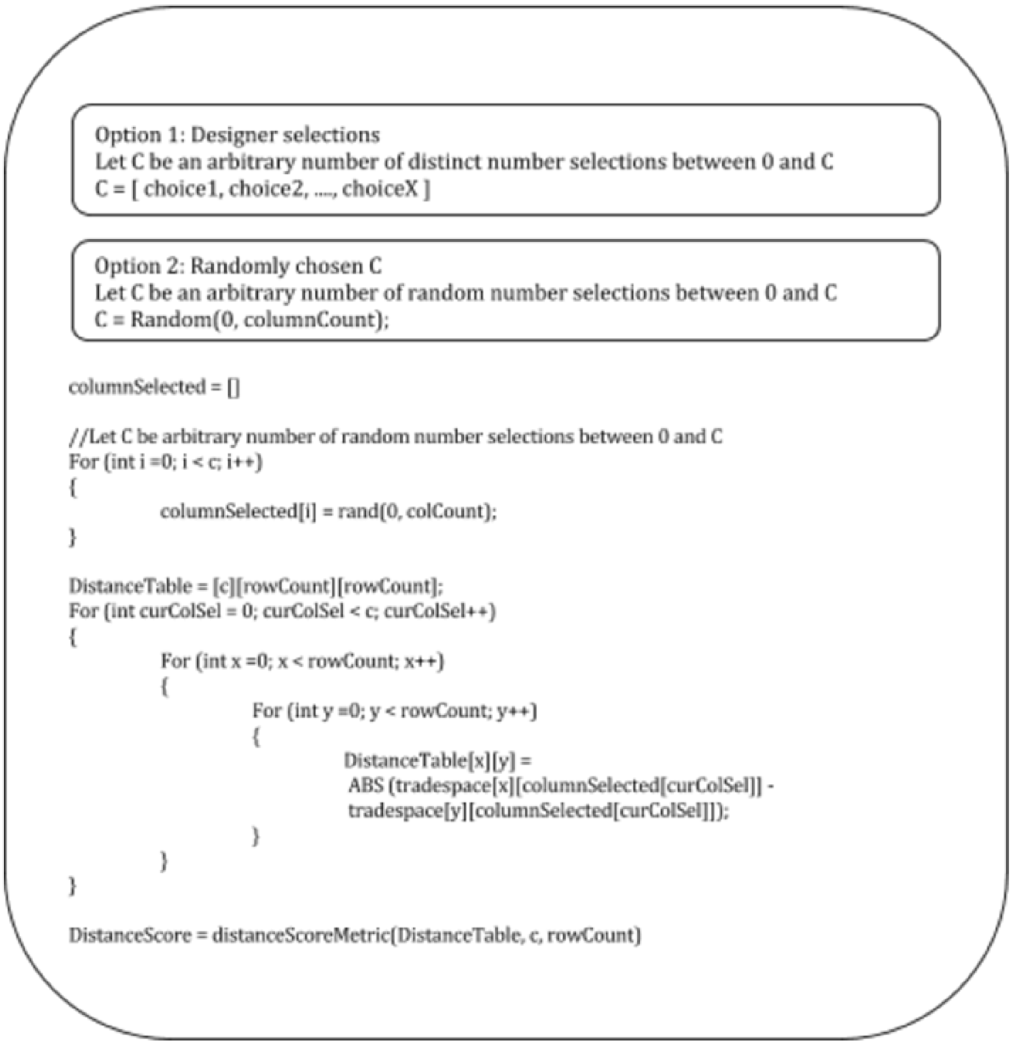
Option 2: Randomly chosen for SelectedCol
Let C be an arbitrary number of random number selections between 0 and C
SelectedCol = Random(0, columnCount);

```
float[] SingleDistance( float[][] dataTradespace, int selectedRow, int selectedCol) {
    float [] distanceTradespace = new float[numRows];
    columnSelected = [];

    //Let C be arbitrary number of random number selections between 0 and C
    for (int i =0; i < c; i++)
    {
        columnSelected[i] = rand(0, colCount);
    }

    for (int curColSel = 0; curColSel < c; curColSel++) {
        for (int x =0; x < rowCount; x++) {
            for (int y =0; y < rowCount; y++){
                distanceTradespace[x][y] = getDistance(
                    dataTradespace[selectedRow][selectedCol],
                    dataTradespace[curRow][selectedCol]);
            }
        }
    }
    return distanceTradespace;
}

float getDistance (float A, float B){
    return Math.Sqrt ((A - B) ^ 2);
}
```

Figure 11. Multi Column Individual Storage Distance

Combined Multi-Distance

Combined multi-distance is a method of storing a single distance value based on the comparison of the distance between column values in multiple columns.



```
Option 1: Designer selections
Let C be an arbitrary number of distinct number selections between 0 and C
C = [ choice1, choice2, ...., choiceX ]

Option 2: Randomly chosen C
Let C be an arbitrary number of random number selections between 0 and C
C = Random(0, columnCount);

columnSelected = []

//Let C be arbitrary number of random number selections between 0 and C
For (int i =0; i < c; i++)
{
        columnSelected[i] = rand(0, colCount);
}

DistanceTable = [c][rowCount][rowCount];
For (int curColSel = 0; curColSel < c; curColSel++)
{
        For (int x =0; x < rowCount; x++)
        {
                For (int y =0; y < rowCount; y++)
                {
                        DistanceTable[x][y] =
                         ABS (tradespace[x][columnSelected[curColSel]] -
                         tradespace[y][columnSelected[curColSel]]);
                }
        }
}

DistanceScore = distanceScoreMetric(DistanceTable, c, rowCount)
```

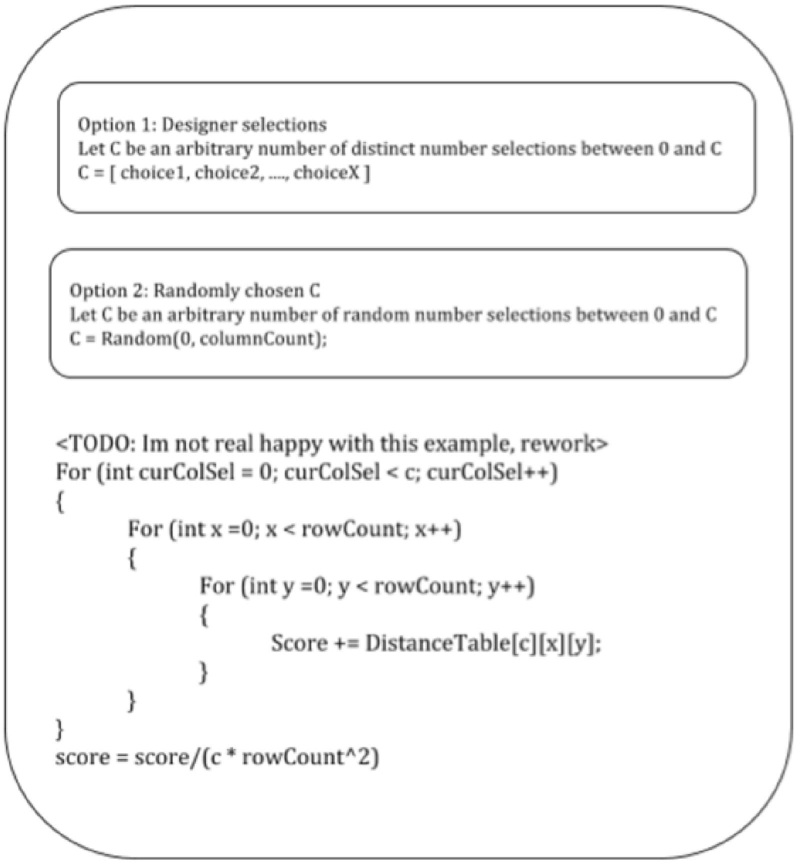
Figure 12. Multi Column Score Random Selection

Methods for Calculating Score Distance

Now that methods have been established for calculating the Euclidean distance in between points in one or more columns, and methods for storing those calculated distances, it is now time to introduce methods for calculating the score of the distance measurements is called distanceScoreMetric. The distanceScoreMetric function can be calculated using multiple methods found below.

Average Closest Selected Column Score

Average closest selected column score is a distanceScoreMetric that is calculated by sorting the table of distances and taking the average distance score for each row based on a chosen number of closest points.



Option 1: Designer selections
Let C be an arbitrary number of distinct number selections between 0 and C
C = [choice1, choice2, ..., choiceX]

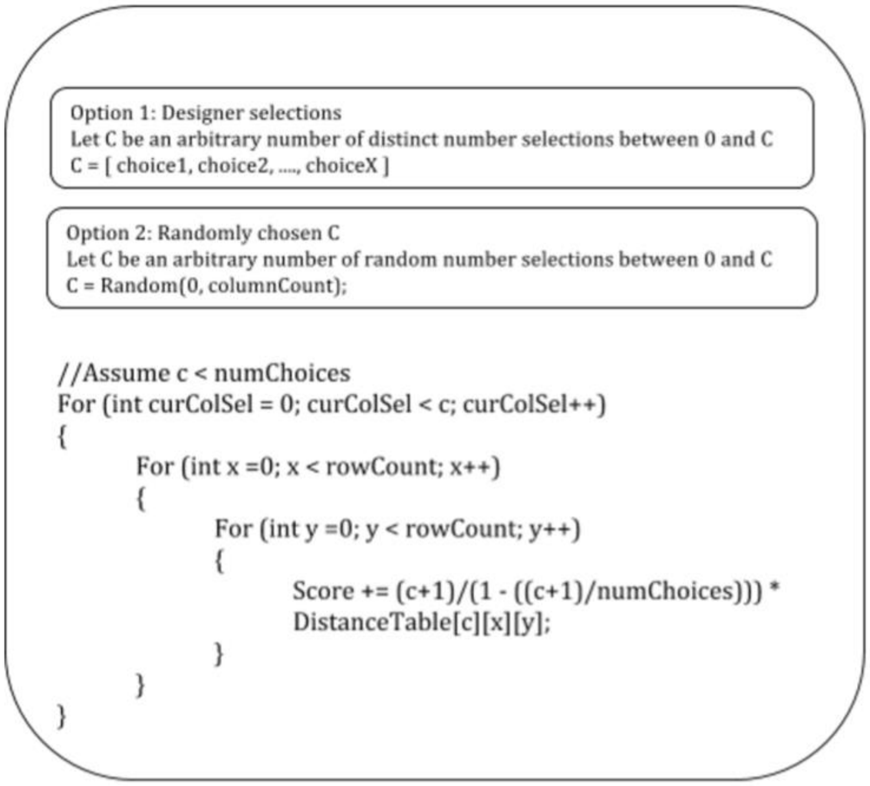
Option 2: Randomly chosen C
Let C be an arbitrary number of random number selections between 0 and C
C = Random(0, columnCount);

```
<TODO: Im not real happy with this example, rework>
For (int curColSel = 0; curColSel < c; curColSel++)
{
        For (int x =0; x < rowCount; x++)
        {
                For (int y =0; y < rowCount; y++)
                {
                        Score += DistanceTable[c][x][y];
                }
        }
}
score = score/(c * rowCount^2)
```

Figure 13. Average curColSel

Weighted Sum Distance Score

Weighted sum distance score is calculated by either randomly selecting or having the designer select columns and assigning a weighting value to those columns. The points are now sorted in ascending or descending order based on designer preference. Choose a number of closest points and multiply or add the distance score for each row based on the selected number of closest points.



Option 1: Designer selections
Let C be an arbitrary number of distinct number selections between 0 and C
C = [choice1, choice2, ..., choiceX]

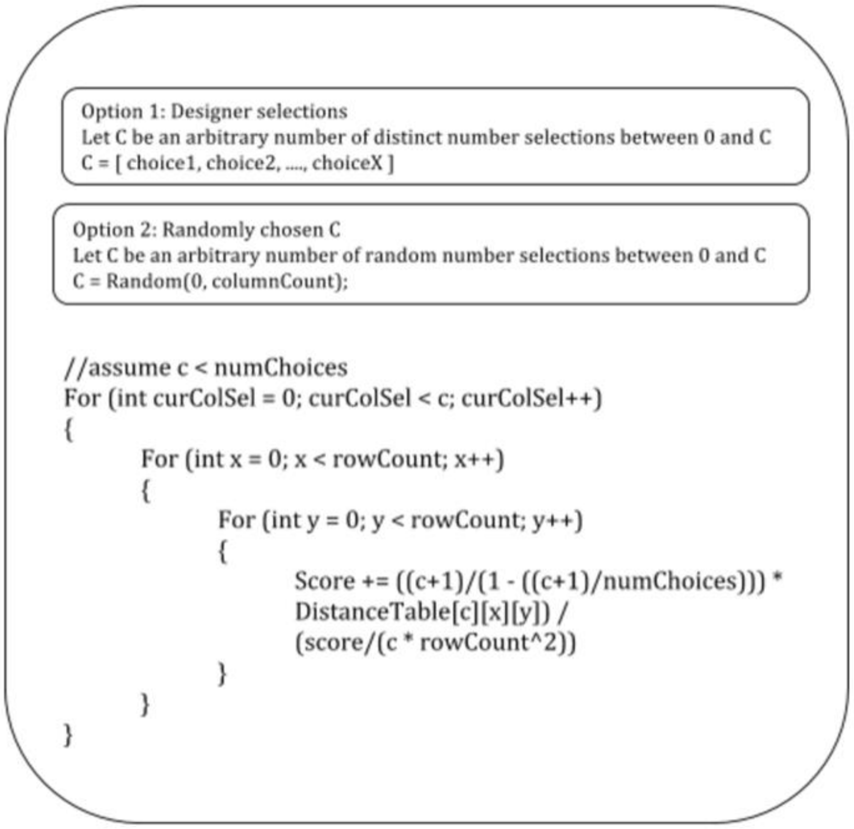
Option 2: Randomly chosen C
Let C be an arbitrary number of random number selections between 0 and C
C = Random(0, columnCount);

```
//Assume c < numChoices
For (int curColSel = 0; curColSel < c; curColSel++)
{
        For (int x =0; x < rowCount; x++)
        {
                For (int y =0; y < rowCount; y++)
                {
                        Score += (c+1)/(1 - ((c+1)/numChoices))) *
                        DistanceTable[c][x][y];
                }
        }
}
```

Figure 14. Method 2 – Weighted Sum

Average Weighted Sum Distance

Average weighted sum distance is calculated by either randomly selecting or having the designer select a chosen number of columns. Sort the table of distances and multiply or add the weighted sum value for each column to every row. Take the average distance score based on the distance between each row and its chosen number of closest to points.



```
Option 1: Designer selections
Let C be an arbitrary number of distinct number selections between 0 and C
C = [ choice1, choice2, ...., choiceX ]

Option 2: Randomly chosen C
Let C be an arbitrary number of random number selections between 0 and C
C = Random(0, columnCount);

//assume c < numChoices
For (int curColSel = 0; curColSel < c; curColSel++)
{
        For (int x = 0; x < rowCount; x++)
        {
                For (int y = 0; y < rowCount; y++)
                {
                        Score += ((c+1)/(1 - ((c+1)/numChoices))) *
                        DistanceTable[c][x][y]) /
                        (score/(c * rowCount^2))
                }
        }
}
```

Figure 15. Average Weighted Sum

A designer has many options to use while searching for a robust design when calculating the distance in between points, storing the results of the distance storing the results of the distance calculations, and using the stored results for calculating a score metric value.

The value returned from the score metric is dependent on the methods the designer chose during the distance calculations and the storage of those results. It would be considered good practice for the designer to try multiple methods and examining the results of each combination of methods chosen as part of the search for a robust design.

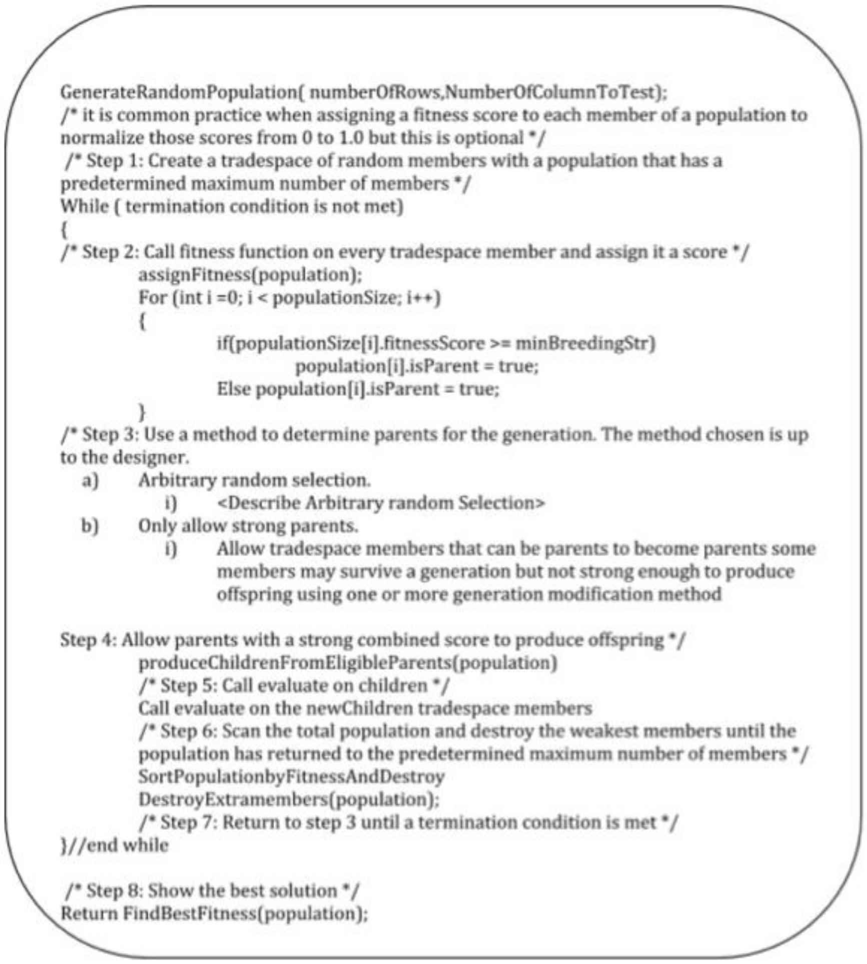
Genetic Algorithm Test Metric for Robustness. Genetic algorithm test metric for robustness takes the concept of the basic Boolean test and expands upon this method by applying the Boolean test to genetic algorithm concepts. The basic Boolean test is applied to a genetic algorithm by observing the value of the fitness score to determine if the fitness score is below, above or in an acceptable value range. If the value of the fitness score passes the Boolean test then the score metric is used to store the number of times a design passes or fails these tests. A design that passes the Boolean test more often is more robust.

The general concept for the genetic algorithm test metric for adding robustness is to create a tradespace of random members with a predetermined max number of members, or it is also acceptable to use a previously created tradespace. Next, create a function that gives an idea on the strength of the members of the tradespace and call this function the fitness function. Perform some action on the parameter values for each member of the tradespace an arbitrary number of times.

In order to use a genetic algorithm when searching a robust design, a designer must understand the basic practices for genetic algorithms such as linear normalization (normalization over the range of 0 to 1.0) and duplicate handling (allow duplicate or force unique column configurations) could be considered when searching for a robust design. Also, methods for determining which members of the population survive to the next generation such as elitism, crossover, and mutation should be implemented in order to ensure efficient use of genetic algorithms in the search for robustness in MOO. Also, ensure that in every generation an action is taken that results in changes to the fitness function value. After the action is performed that results in a change to the function value, perform a test that determines if the data member survives, dies, or is allowed to reproduce new data members.

Genetic Algorithm Example

Multiple genetic algorithm examples are available for use in scientific computing. The following figure provides a basic understanding of how to code a genetic algorithm for use in searching for robust designs.



```
GenerateRandomPopulation( numberOfRows,NumberOfColumnToTest);
/* it is common practice when assigning a fitness score to each member of a population to
normalize those scores from 0 to 1.0 but this is optional */
 /* Step 1: Create a tradespace of random members with a population that has a
predetermined maximum number of members */
While ( termination condition is not met)
{
/* Step 2: Call fitness function on every tradespace member and assign it a score */
          assignFitness(population);
          For (int i =0; i < populationSize; i++)
          {
                    if(populationSize[i].fitnessScore >= minBreedingStr)
                              population[i].isParent = true;
                    Else population[i].isParent = true;
          }
/* Step 3: Use a method to determine parents for the generation. The method chosen is up
to the designer.
   a)     Arbitrary random selection.
              i)     <Describe Arbitrary random Selection>
   b)     Only allow strong parents.
              i)     Allow tradespace members that can be parents to become parents some
                     members may survive a generation but not strong enough to produce
                     offspring using one or more generation modification method

Step 4: Allow parents with a strong combined score to produce offspring */
          produceChildrenFromEligibleParents(population)
          /* Step 5: Call evaluate on children */
          Call evaluate on the newChildren tradespace members
          /* Step 6: Scan the total population and destroy the weakest members until the
          population has returned to the predetermined maximum number of members */
          SortPopulationbyFitnessAndDestroy
          DestroyExtramembers(population);
          /* Step 7: Return to step 3 until a termination condition is met */
}//end while

 /* Step 8: Show the best solution */
Return FindBestFitness(population);
```

Figure 16. Sudo Code Example For a Genetic Algorithm

Using the basic concept of a genetic algorithm, a designer has access to a powerful tool that can be used for calculating the score metric value used when searching for a robust design. The designer is able to analyze many different parameter values combinations which provides opportunity to observe the results of modifying parameter values on the score metric. Observing the results of the changes displayed by the score metric can provide a good understanding of the robustness of a design.

Combined Test Metric for Robustness. Combined test metric for robustness uses a combination of methods from two or more robustness tests to create a higher level robustness score. An example of such a combination of methods would be combining the robustness score of the distance and genetic algorithm robustness tests. As each method for testing robustness has the potential for being computationally expensive, it is recommended to take caution to keep the total computation time within an acceptable range for your tests.

There are many possible tests for finding robustness. The Boolean test, distance test, genetic algorithm test and the combined test metric have been listed here but there are many more known and undiscovered methods for finding robustness. There is no known best method for finding robustness so the best option for a designer is to ready multiple methods for searching for robustness, and to apply them as interchangeable modules.

We looked at adding robustness as a percentage. We then described the Boolean tests for calculating robustness such as the Boolean test and the different components of the more complicated distance test. We also described a genetic algorithm test and briefly explained that it is an option for the designer to combine testing methods. By understanding multiple means for searching for design robustness, a designer has more control over understanding the ability of a design to withstand changes throughout a design’s lifecycle.

CHAPTER III - METHODOLOGY

Multi-objective optimization (MOO) is a vast field of study applied to multiple areas of scientific study where tradeoffs among competing interests must be balanced and considered. Robust design adds an additional layer of analysis to MOO trying to find advantageous tradeoffs among competing interests where there is uncertainty and decision-makers seek a robust solution that will still be acceptable even with expected variance in outcomes. In this work, we utilize the concept of parameter variance from multi-objective optimization in order to search for a robust design within the motivating Small Surface Combatant Task Force (SSCTF) dataset (please see the introduction for an overview of the SSCTF dataset and related project). While the research was focused on the SSCTF dataset, the methods presented herein are applicable to wide variety of multi-stage design and decision problems. The SSCTF dataset was utilized to show that the concepts within this work had real world application and could be utilized to extend and improve a State of the Art design process.

Two new methods of analysis to estimate design robustness are developed when exploring the complex relationships between design parameters, metrics, and models applied to the SSCTF dataset (explained further within the section). These two methods of analysis are developed while using the SSCTF metrics and models to estimate design robustness. In summary, the philosophy of this work was to utilize the SSCTF dataset and its metrics to show real world application of a new set of robustness estimate methods and this work also focused on ensuring transferability of this methodology to alternative datasets and problems.

Research Approach

The approach of this work was to provide a two-step process in which step 1 seeks to break down the concepts required for understanding the components of our multi-objective optimization problem and step 2 focused on providing 2 algorithms used in exploring the tradespace for a robust design estimate given uncertain changes to parameter values. While performing step 1 and step of the approach, effort was taken to ensure that transferring application of the methods to alternative data sets was intuitive.

Step 1: The Three Concept Levels Method for Deconstruction of a Multi-Objective Optimization Problem

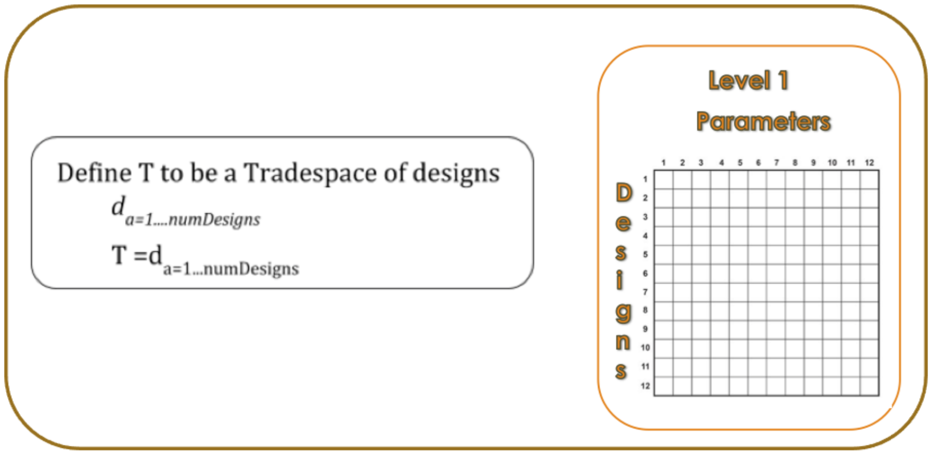
The First Concept Level

TRADESPACE
LEVEL 1

Figure 17. The first concept level

The first concept level of a multi-objective optimization problem is the tradespace. The tradespace is the most basic component for the multi-objective optimization problem. The tradespace is composed of designs and deconstruction of the tradespace is a beginning point for components of a multi-objective optimization problem. In general, a tradespace is composed of variables that represent the capabilities of designs. These variables also provide insight into the relationships between the variables and are utilized in some combination to provide the designer a means for comparing the effectiveness of the designs.

Tradespace Components. The SSCTF tradespace was a complex configuration of design parameters which provided a description of the capabilities of a ship design. The SSCTF tradespace consists of several different components which are described below and are useful in understanding the tradespace utilized in this work. The first component is non-numeric designations which are the ID descriptions of the different designs. These ids allow the designer to identify the categorical capabilities of the ship design which are the family, combat capability, and combat capability alternative for a design. The family of the design is based on the HM&E or hull mechanical and engineering configuration of a ship design. The combat capability is the type of warfare the ship is designed to handle such as reconnaissance or anti-submarine warfare. The combat capability alternative is a variation of a combat capability that is capable of handling a different type of warfare than the original combat capability. The next component of the tradespace is the composed of fixed ship design properties. These properties are design inputs such as the length of the ship or the type of radar the ship utilizes. Next, we have modeled design outputs. Modeled design outputs are properties of the design that are directly affected by design inputs. An example would be the weight of the ship which is affected by many inputs such as the length of the ship, or the number and size of the weapons placed on the ship.

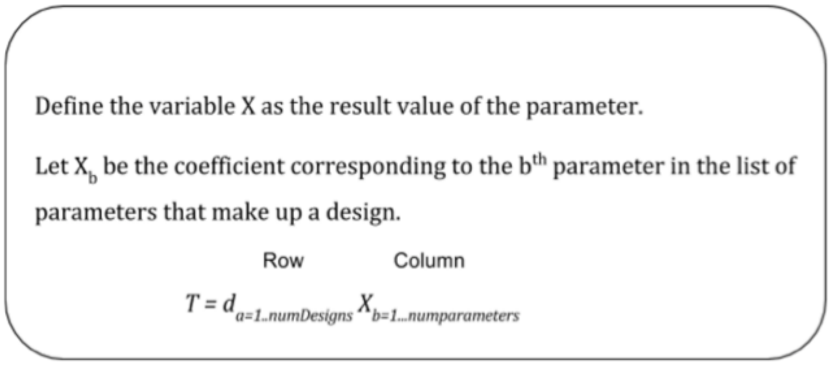

Define T to be a Tradespace of designs

$d_{a=1....numDesigns}$

$T = d_{a=1...numDesigns}$

Figure 18. Definition of a tradespace

Each design in the tradespace is represented by a row of data. The rows of data in the tradespace are composed of columns that are the *parameters* that make up a design. Each parameter is defined here as the numerical representation of the level of contribution provided to the multi-objective optimization problem. The numerical value representing a parameter can be the result of an equation or simply a static number.


Define the variable X as the result value of the parameter.

Let X_b be the coefficient corresponding to the bth parameter in the list of parameters that make up a design.

$$\begin{array}{cc} \text{Row} & \text{Column} \\ T = d_{a=1..numDesigns} & X_{b=1...numparameters} \end{array}$$

Figure 19. Parameter Definition

For each of the X_b designs, there exists a range of possible values. This range of values can be most easily understood as a range of values between a minimum value and a maximum value.

$$T = d_{a=1\ldots numDesigns} + X_{b=1\ldots numParamters,}$$
$$X_b = f(minVal, maxVal)$$

Figure 20. Parameter Value Range Definition 1 of 2

Because an infinite number of values that can be represented between any two numbers, a designer must use a value that represents a meaningful change in the design as the distance in between any two points in the range of values for each parameter.

Redefine X_b to more accurately represent the range of
values available

$$T = d_{a=1\ldots numDesigns} + X_{b=1\ldots numParamters,}$$
$$X_b = f(minVal, maxVal, stepSize)$$

Figure 21. Parameter Value Range Definition 2 of 2

The Second Concept Level

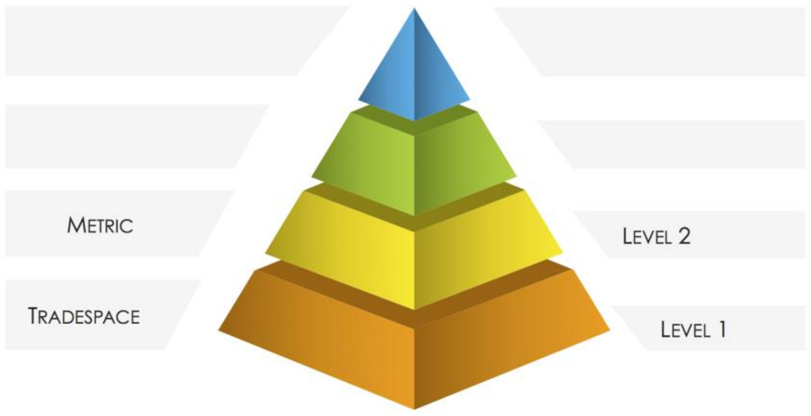


Figure 22. The second concept level

The second concept level of a multi-objective optimization problem can be understood as additional columns of data added to the tradespace. These additional columns are defined here as metrics. Metrics are additional parameters added to a tradespace that are the result of functional calculations on the tradespace. For example, any algorithmic combination of parameter column values would be acceptable for creating a metric. Essentially, a metric is a meaningful calculation that the designer can use to show relationships between column values.

Let $M_{c=0....numMetrics}$ represent the set of additional metrics:

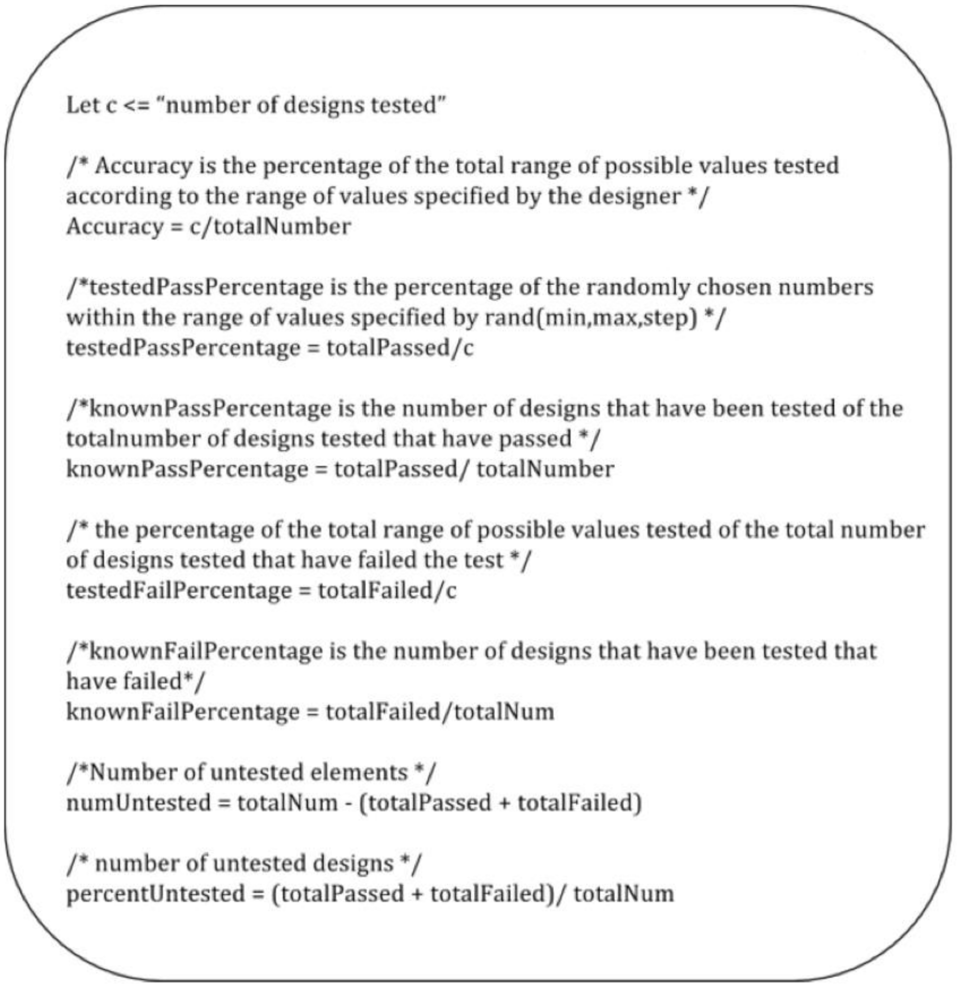
$T = d_{a=1...numDesigns} + X_{b=1...numParamters+M_{c=0...numMetrics}}$

$X_b = f(minVal, maxVal, stepsize)$

Figure 23. Additional Metrics

Metrics are commonly created in three different ways; a static metric is a number that is not calculated; independent metric is calculated using an algorithm that doesn’t rely on any other metric to obtain a value; and dependent function metrics which are calculated using an algorithm that relies on other metrics to obtain a value. Metrics added to the tradespace should provide a meaningful way for the designer to better understand both the relationships between parameters and provide insight into the operational effectiveness of a design. A list of pseudo code examples of metrics can be observed in the table below.

Metric Value Definitions



```
Let c <= “number of designs tested”

/* Accuracy is the percentage of the total range of possible values tested
according to the range of values specified by the designer */
Accuracy = c/totalNumber

/*testedPassPercentage is the percentage of the randomly chosen numbers
within the range of values specified by rand(min,max,step) */
testedPassPercentage = totalPassed/c

/*knownPassPercentage is the number of designs that have been tested of the
totalnumber of designs tested that have passed */
knownPassPercentage = totalPassed/ totalNumber

/* the percentage of the total range of possible values tested of the total number
of designs tested that have failed the test */
testedFailPercentage = totalFailed/c

/*knownFailPercentage is the number of designs that have been tested that
have failed*/
knownFailPercentage = totalFailed/totalNum

/*Number of untested elements */
numUntested = totalNum - (totalPassed + totalFailed)

/* number of untested designs */
percentUntested = (totalPassed + totalFailed)/ totalNum
```

Figure 24. Sudo Code for Commonly useful Metric Value Examples

The Third Concept Level

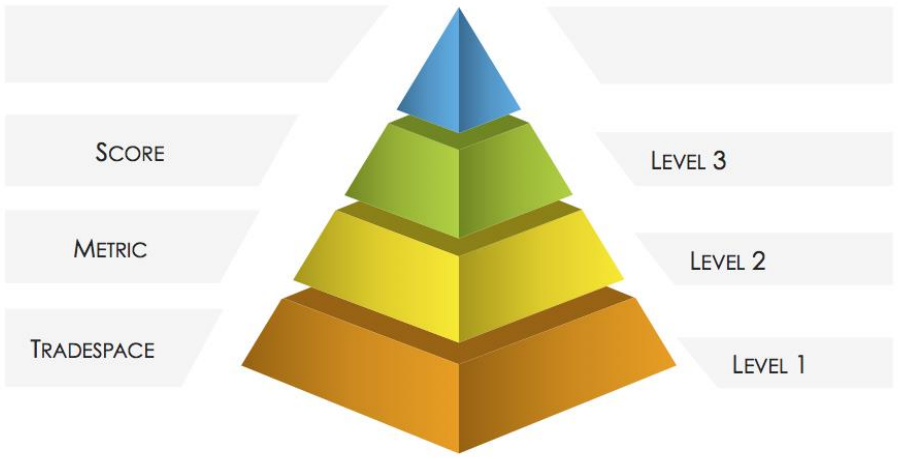


Figure 25. The third concept level

A metric is a value that represents the result of a functional combination of parameters and is included in the list of parameters within the design tradespace. The most important metric in multi-objective optimization problems is the score metric. The score metric represents the value of a design and is the core component of the *The Third Concept Level*. To give a little better understanding of what is represented by a score metric, a score metric is not limited to but could represent any of the following things: monetary value, level of effectiveness of a group of parameters, percentage of capability.

In general, multi-objective optimization can be thought of as a tradespace composed of designs consisting of a set of parameters each of which represent a range of possible values with a determined distance between each point. It is common practice to add metrics to the list of parameters in order to show relationships between parameters, but there is one metric that is more critical than the other metrics. This critical metric represents the third concept level of multi-objective optimization and is used to represent the expected level of performance of a design. We call this critical metric the score metric. The score metric is important because it represents the way a designer can compare one design to other designs. It can also be used to determine whether or not a design passed testing.

Let S represent the *score metric:*

$$T = d_{a=1\ldots numDesigns}{}^{+\, Xb=1\ldots numParamters\, +\, M_{c=0\ldots numMetrics} +\, S}$$

$$X_b = f(minVal, maxVal, stepsize)$$

Figure 26. Score Metric

As introduced in concept level 2, there are three ways in which metrics are calculated and those methods are static, independent, and dependent. Since the score metric is critical in gauging ability of a design to perform, it is important to choose the right method when calculating the value of the score metric.

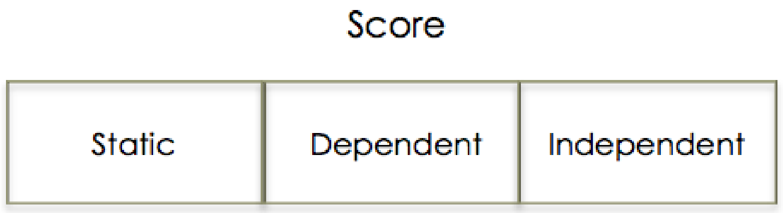


Figure 27. Three different types of score metric

Of the methods for calculating a metric (static, independent, and dependent), the score metric should never be calculated using a static number. A static number would imply the designer already knew whether or not a design passed testing and how well the design performed before testing. The score metric should also not be an independent function as being an independent function would mean that changes made to the parameters of a design would have no impact on the score metric value. The score metric should be a dependent metric function that relies on the parameter values and possibly other metric values for gauging one design's effectiveness against another design.

Summary: The General Description of the first three concept levels of Multi-Objective Optimization Problem

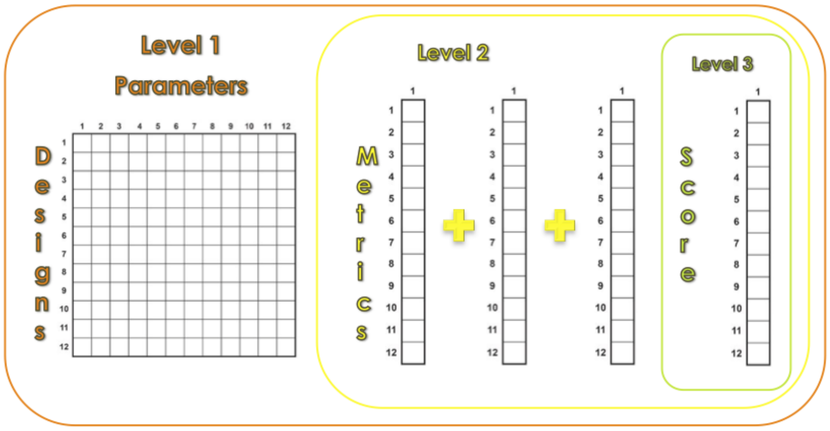
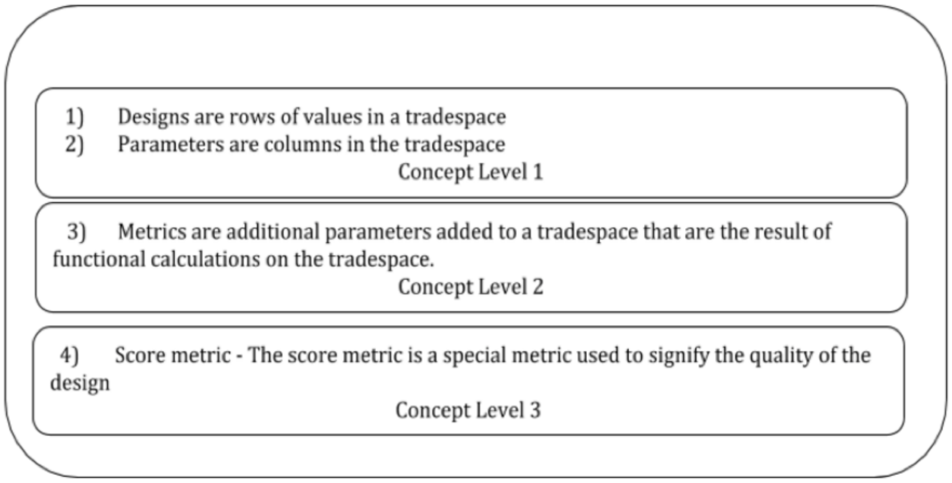


Figure 28. Three levels of capability concept

In summary, the first three concept levels of a multi-objective optimization are the basic concepts required for understanding an approachable mechanism for deconstruction of a multi-objective optimization problem. Descriptions of the first three concept levels of a multi-objective optimization problem were explained as the tradespace, metric, and score metric concept levels.



1) Designs are rows of values in a tradespace
2) Parameters are columns in the tradespace
Concept Level 1

3) Metrics are additional parameters added to a tradespace that are the result of functional calculations on the tradespace.
Concept Level 2

4) Score metric - The score metric is a special metric used to signify the quality of the design
Concept Level 3

Figure 29. Summary of concept levels of a multi Objective optimization Problem

An understanding of these three concepts levels provides the designer a foundation needed for adding a fourth concept level and the focus topic of the next section, adding robustness to multi-objective optimization.

Fourth Concept Level: Adding Robustness to Multi-Objective Optimization

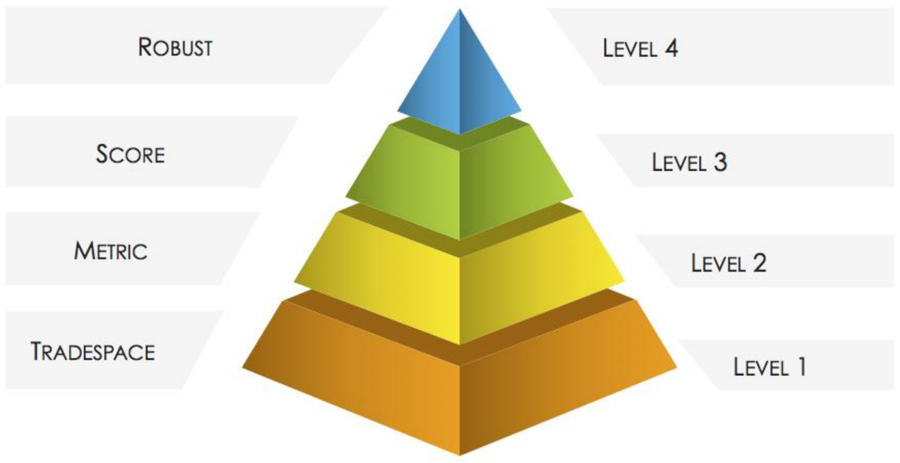


Figure 30. Introduction to adding robustness

We have established a working description of the three concept levels for a multi-objective optimization problem and we now need to look at the additional requirements that are needed for adding the fourth concept level of a multi-objective optimization problem. The fourth concept level is adding robustness to a multi-objective optimization tradespace.

Adding Robustness by Testing Changes to the Score Metric

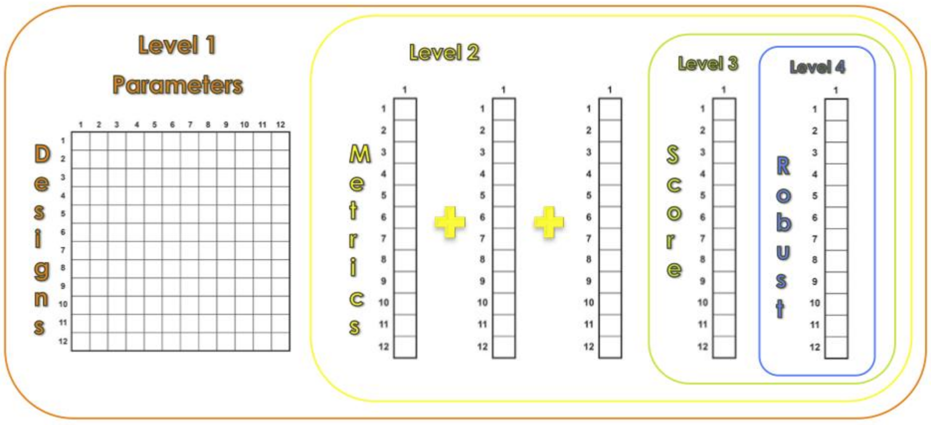


Figure 31. Four concept levels of multi-objective optimization

Adding robustness to a multi-objective optimization problem is not a trivial problem as it requires additional computation and understanding of the solution space of designs. The *robustness score* is a metric that provides a numerical representation describing a design's ability to withstand change. The *robustness score* or *R* is calculated by measuring the effect of changing the value of a parameter used in calculating the *score metric (S)* for a design. As we recall, the score metric is used for describing a design's ability to perform and a means for comparing a design to other designs. By measuring the change in the value of the score metric, we observe three of the possible types of robustness metrics. *Positive acceptable robustness* is a scenario in which the robustness metric only accepts score metric values that are better than the original design's score. An *indifferent acceptable robustness* is a type of robustness metric in which the robustness metric accepts *score metric* values that are better or worse than the original score metric value with the condition that the score metric value must be an acceptable design. A *negative acceptable robustness* is a scenario that only accepts score metric values that are lower than the original score metric value. The focus of this work will utilize the *indifferent acceptable score metric* scenario.

$$d_{a=1\ldots numDesigns} + X_{b=1\ldots numParamters} + M_{c=0\ldots numMetrics} + S + R_{d=0\ldots numRobustScore}$$

$$X_b = f(minVal, maxVal, stepsize)$$

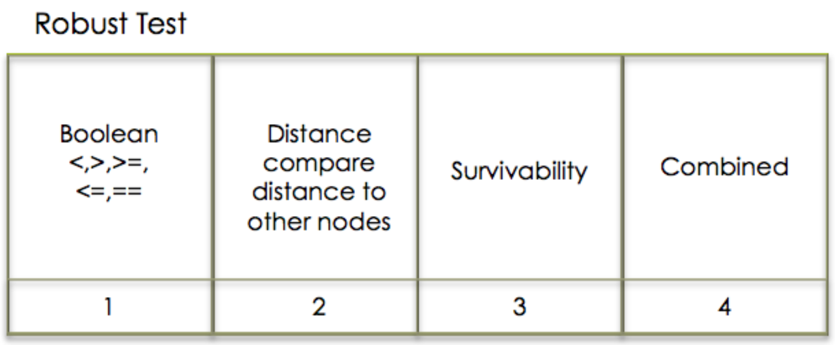
Figure 32. Adding robust score to the tradespace

Robust Score Metric as a Percentage. Positive, indifferent, and negative are the types of robustness tests for the score metric that have been described. Understanding these three methods is essential to understanding how to calculate the robust score metric. However, these three methods will only result in a True or False answer. Having True or False does not fulfill the requirement of a numerical value representation of the level of robustness. While it is acceptable to consider True or False be equal to 1 and 0 respectively, assigning the numerical representation of True or False to the robustness metric is insufficient. It is unlikely to locate a design that is fully robust, which in this case would be represented by a 1. A fully robust design would be able to withstand any changes to its parameter values and still be able to fulfill the required capabilities of the design, which is an unlikely scenario in product design.

It is more likely to locate a design that is able to pass the robust test a percentage of the time. By listing resiliency as a percentage, a designer can expect a design to withstand a change and remain feasible a percent of the time which is valuable information. In addition to being able to recognize a design's ability to withstand changes a percent of the time, assigning the robustness score as a percentage allows the designer a useful means for being able to compare designs and also the ability to reasonably predict the failure rate for a design.

Modifying a Value for Finding a Robust Percentage. In order to calculate the percentage score that represents robustness, we need to modify a parameter value that is used in calculating the score value for a design. Modifying a parameter can be as simple as replacing the value of a parameter with another possible value within the predetermined range of possible values for a parameter. Modifying a parameter can also be a more complicated process of performing a calculation to assign a new parameter value. After we modify a parameter value, we need to perform one of three tests to determine whether or not the score metric is within range of acceptable values as determined by the robustness testing scenario chosen by the designer for an acceptable design. For example, in order to fulfill the requirements of the indifferent acceptable score metric, the resulting value of the calculation would need to lie on or in between the max and min possible values for the parameter.

Test for Calculating Robustness. There are many possible tests for robustness and since there is no known best method for testing design robustness, it is best for the designer to understand at least a few different types of robustness tests. In order to limit the potential problem space for different types of robustness tests, this work is going to focus on the Boolean test, however, three additional robustness testing methods are explained in the background section to provide the reader with addition insight into options for calculating robustness. A detailed description of our described Boolean test for robustness is now provided.



Robust Test

Boolean <,>,>=, <=,==	Distance compare distance to other nodes	Survivability	Combined
1	2	3	4

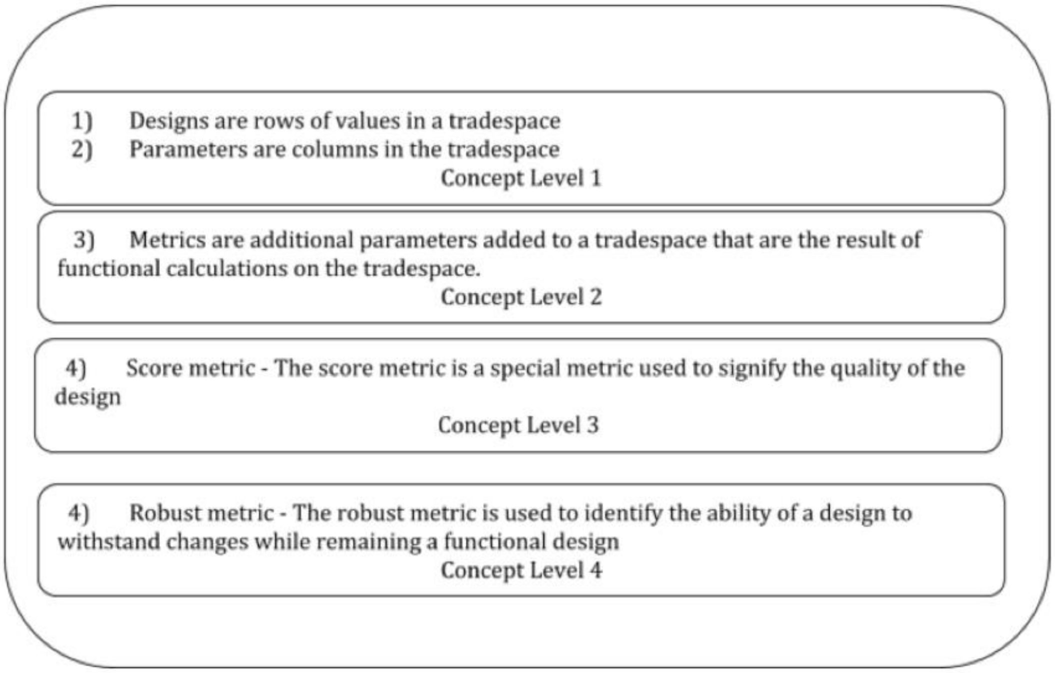
Figure 33. Robustness Tests

Basic Boolean Test Metric for Robustness. The basic test metric for robustness of design begins with a Boolean test in the form of A(<, >, <=, >=, ==, !=)B. Multiple test can be linked together when calculating the Boolean answer. The links between tests can be represented by using linking terms such as ‘and’, ‘or’

a) A (<,>,<=,>=,==,!=)B and C(<,>,<=,>=,==,!=)D
   1) A > B and C == D
   2) A < B or D != C
   3) A > B and C == D or A == C

There are many possible tests for finding robustness. The Boolean test, distance test, genetic algorithm test and the combined test metric have been described but there are many more known and undiscovered methods for finding robustness. There is no known best method for finding robustness so the best option for a designer is to ready multiple methods for searching for robustness, and to apply them as interchangeable modules.

Summary of Adding the Robust Metric to a Multi-Objective Optimization Problem


1) Designs are rows of values in a tradespace
2) Parameters are columns in the tradespace
Concept Level 1

3) Metrics are additional parameters added to a tradespace that are the result of functional calculations on the tradespace.
Concept Level 2

4) Score metric - The score metric is a special metric used to signify the quality of the design
Concept Level 3

4) Robust metric - The robust metric is used to identify the ability of a design to withstand changes while remaining a functional design
Concept Level 4

Figure 34. Summary of the concept levels of a multi-objective optimization problem with robustness

Step 2 Part 1: Permutation Stability Analysis -

Calculating Robustness with Substitution

Introduction: Substitute Primary Parameter Value from Feasible Design with Primary Parameter Value from Another Design

Our search for a robust design began within a tradespace of feasible and infeasible naval ship designs. Feasible designs are the designs which provided an acceptable score metric value above the predetermined threshold of -1. The SSCTF design team provided the threshold score metric value. The infeasible designs are described to be any design that did not possess a score metric value above -1. During initial testing of the design space, permutation testing was performed utilizing both feasible and infeasible designs. It was realized that permutation testing performed on an infeasible design almost always resulted in a failed final design. In order for a design to pass testing, a design had to pass all 16 testing metrics. Because of having to pass all 16 testing metrics, it is unlikely that modifying a single design parameter on a failed design would affect the outcome of the testing metrics because of the complexity of the relationships between design parameters. It should be noted that it is possible for a design to fail initial testing and still pass future score metric testing, however, further testing of a design that fails initial score metric testing may result in simply determining that the design was infeasible and that further testing of this infeasible design could have been better utilized by testing a starting viable design.

The tradespace of designs can be divided into 4 areas based on the likelihood of success and the ability of a design to withstand testing as observed in figure 35 below. The first area of the design space that we are going to discuss is the infeasible and non-resilient area. These are the designs that are both incapable of performing all required design tests and unable to withstand changes to parameter values. The infeasible and non-resilient area of the tradespace is the worst-case scenario. Next, we have the infeasible but resilient area of the tradespace. These are the designs that do not pass all required testing of the tradespace but are able to withstand changes to parameter values without much change to design performance. Next, we have the non-resilient but feasible are of the tradespace. These designs are able to accomplish the required area task of the design but are unable to withstand changes to parameter values. This area of the design space is where many final versions of designs are located and is a leading reason for research into multi-objective optimization. The non-resilient feasible designs are often the optimal designs within the tradespace. What this means is that these designs outperform all other designs within the tradespace but are unable to withstand changes to design parameters. The final area of the tradespace, and the most desired outcome is the area of the tradespace known as the feasible and resilient area of the tradespace. This area of the tradespace is the area in which designs are both capable of performing all required tasks of a design and the designs are also able to withstand reasonable changes to design parameters. An optimal feasible resilient design is the most ideal case of this scenario, however, locating such a design may not be possible so the alternative of a design that is both feasible and resilient but may not be the optimal design is also desired.

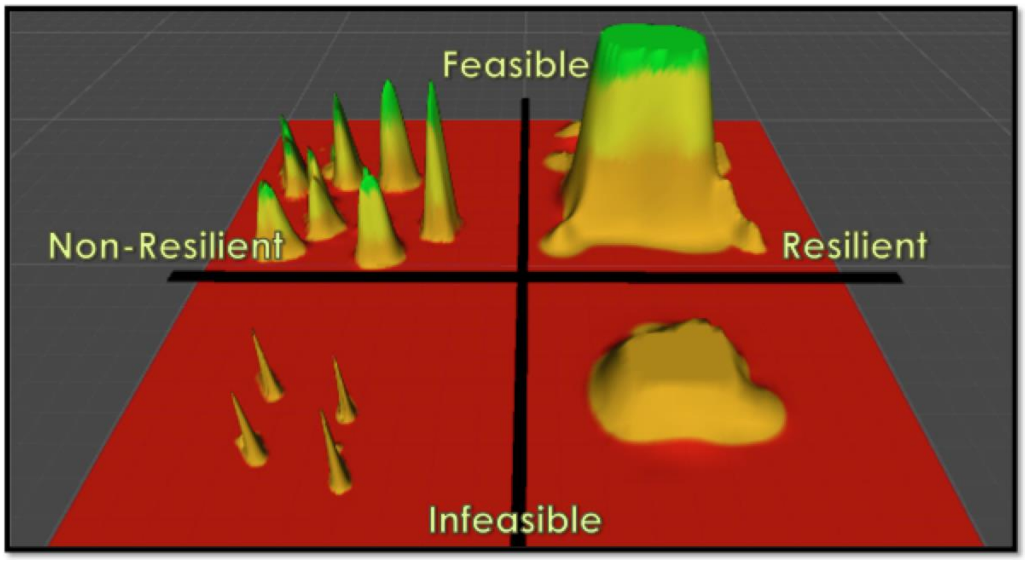


Figure 35. Four regions of a design tradespace

Description of Permutation Testing. Permutation analysis is a multi-step process for assisting a designer in selecting a design with a percentage level of resistance to changes in a design's parameter values. We begin the description of permutation analysis by identifying critical parameters that have the largest impact on the score metric value of a design. For this work, we focused on Free Space, Free Weight, Free Power, and Free Cooling of a naval ship design. These 4 critical variables were predetermined by the

SSCTF design team as the 4 parameters that had the largest impact on the likelihood that a design would pass the 16 testing metrics with a score metric value greater than -1.

Permutation analysis, as applied to the SSCTF dataset, began by dividing the tradespace up into the 5 different mechanical model families. These mechanical model families are known as I1, I2, IC, M1, M2 (described in Introduction). For each mechanical model, we repeat the substitution analysis method for each of the 4 critical parameter values. The following is a description of permutation analysis as applied to a single critical parameter. This method was applied to each of the 4 critical parameter values.

Permutation Analysis for a Single Critical Variable. To begin permutation analysis start by storing all available values for a critical variable using the designs in the tradespace from a mechanical model into a data structure. Next, randomly chose a target design and trade the value of another design's critical parameter from the data structure of available values with the value of the same critical parameter in the target design. After the value of the target design's parameter has been substituted with the value from the data structure of available values for the selected critical parameter, recalculate the score for the modified initial design. If the design after substitution was performed is no longer within the range of acceptable values as determined by the designer, discard the modified design. Repeat the substitution of the chosen critical parameter value from the target design with all other available values from the data structure of available values for the chosen critical parameter. Next, recalculate the score metric of the target design after every substitution to acquire a total number of feasible designs for the target design. The total number of feasible designs after substitution can be used as the robustness score, or you may perform a calculation based on the likelihood a design is still feasible after substitution. A target design with a high robustness score means that the target design is resilient for the chosen critical variable because the target design can withstand changes to the chosen critical variable and still remain a feasible design.

The method described so far in the description of the substitution method would be able to calculate the robustness score based on one chosen parameter in a design and thus the robustness score would show the ability of the target design to withstand changes for one chosen parameter. In order to determine a more complete robustness score for a design, the substitution process should be repeated for all critical variables. Since the process of substitution is the same for each critical variable, there is good opportunity to run the code in parallel for each of the chosen parameters.

Optimizing the Permutation Analysis Method: Duplicate Tests

At this point in the permutation analysis, one could think about storing the value combinations of the primary variables into an array so that duplicate feasibility tests are not performed. As long as care is taken to ensure that processes are not performing work on the same design using the same value, then testing if a value has been tested before a process uses the value is a small overhead in comparison to allowing duplicate tests.

Subset Testing

If your tradespace contains a large enough number of designs to make permutation analysis computationally infeasible with all parameters in each design, then it is acceptable to perform permutation analysis on a subset of designs from the list of possible designs for each design in the tradespace. Performing permutation analysis on a subset of the possible solutions will obtain a robustness estimate, but you must keep in mind that choosing fewer than all solution values for permutation analysis on target designs could result in less accuracy of the robustness score, and therefore should be listed as approximation of the robustness score.

General Example of Permutation Analysis

Permutation analysis can provide a way to identify designs from the set of feasible designs that are better at withstanding changes to parameter values and are thus more robust. In order to assist a designer in utilizing the permutation method, a generic example of permutation analysis method is now presented.

Step 1: Generate design variation and assign feasibility score

Define T to be a Tradespace of designs $d_{i=1..n}$

T = $d_{i=1..n}$

Each design d_i has properties $(V_{i=1..n}R\ ,\ [X_{i=1..n}\ ])$

V : is a key variable of the design

R : is the range of possible values for a variable

[X] : The list of parameters within the tradespace that do not change

Table 3

Design space: d_i

ID	**V_1R**	**V_2R**	**V_nR**	**[X_1,X_2,X_3]**
1	50-150	30-150	...	1,20,10

$$d_{i=1..n} = V_{i=1..n}R\ ,\ [X_{i=1..n}\ ]$$

Perform Monte Carlo on V_1 in order to select points within the range of possible Values. In this case, the total size of the design tradespace is:

$$(VR)_{size} * (VR)_{size} = (150-50) * (150-30) = 12000_{Tradespace\ Size}$$

12000 designs is not a very large design tradespace, however, if we are dealing with a larger number of Key Variables or larger ranges of values for the key variables, then the potential size of the tradespace grows rapidly. In order to deal with the large numbers of potential designs, sampling methods such as Monte Carlo sampling can be used to help analyze the solution space.

Table 4

Design Space: d_i with variation for each V

ID	**V_1R**	**V_2R**	**[X_1,X_2,X_3]**
1	50	150	1,20,10
2	92	48	1,20,10
3	143	37	1,20,10
4	150	50	1,20,10

For each variant, *V* use a testing method such as the Boolean, distance, genetic algorithm, and combined tests for robustness when determining whether a design generated by the Monte Carlo generation of data is a feasible design.

Boolean testing for feasibility method example

If the value of $|V_1 - V_2| >= 100$ then Feasible

Else Infeasible

F(v) = (Feasible|Infeasible)

Table 5

Design space: d_i with variations for each K and

ID	**V_1R**	**V_2R**	**[X_1,X_2,X_3]**	***F(v) (Feasible| Infeasible)***
1	50	150	1,20,10	Infeasible
2	92	48	1,20,10	Infeasible
3	143	37	1,20,10	Feasible
4	150	50	1,20,10	Feasible

So far the feasibility design score for the base design is 0.5 as 2 of the 4 tested designs are feasible:

Table 6

Design space: d_i with variation for each K and

ID	**V_1R**	**V_2R**	**[X_1,X_2,X_3]**	***Total Feasible Design Score***
1	50-150	30-150	1,20,10	2/4 = .5

Now substitute the values within rows 3 and 4 as they were the feasible designs for the variables from column V_1. Now substitute the parameter values within column V_1, from the designs that were feasible with one another. In this case, we are substituting the values of rows 3 & 4.

Table 7

Design space: d_i with variation for each K

ID	V_1R	V_2R	$[X_1,X_2,X_3]$	***F****(v)* *(Feasible|Infeasible)*
1	50	30	1,20,10	Infeasible
2	92	48	1,20,10	Infeasible
3	150	37	1,20,10	Feasible
4	143	50	1,20,10	Infeasible

After the substitution, row 3 is the only row that remains feasible making the design feasible for three designs out of the 6 designs tested. After the swap of feasible designs, the design remains feasible 50% of the time. In order to increase the rate at which substitution method finds the feasible designs from the range of possible design combinations, during each iteration of swapping values between feasible designs, new values should be generated for all key parameters in all designs that were infeasible.

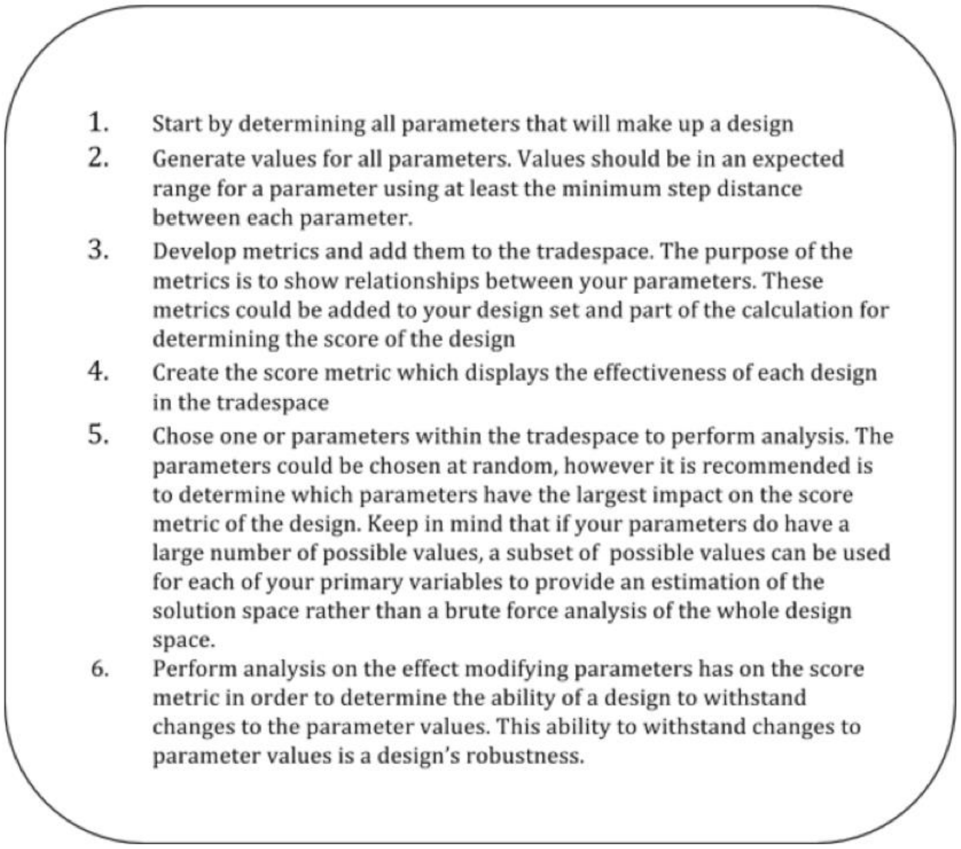

1. Start by determining all parameters that will make up a design
2. Generate values for all parameters. Values should be in an expected range for a parameter using at least the minimum step distance between each parameter.
3. Develop metrics and add them to the tradespace. The purpose of the metrics is to show relationships between your parameters. These metrics could be added to your design set and part of the calculation for determining the score of the design
4. Create the score metric which displays the effectiveness of each design in the tradespace
5. Chose one or parameters within the tradespace to perform analysis. The parameters could be chosen at random, however it is recommended is to determine which parameters have the largest impact on the score metric of the design. Keep in mind that if your parameters do have a large number of possible values, a subset of possible values can be used for each of your primary variables to provide an estimation of the solution space rather than a brute force analysis of the whole design space.
6. Perform analysis on the effect modifying parameters has on the score metric in order to determine the ability of a design to withstand changes to the parameter values. This ability to withstand changes to parameter values is a design's robustness.

Figure 36. General algorithm for locating a robust design through substitution analysis

Sudo Code Example of Permutation Stability Analysis Applied to SSCTF Dataset. Permutation Stability Analysis is the core effort of this work so a sudo code example of the selection of a single design value applied to the SSCTF dataset is provided below. The method begins by randomly selecting a design through the selection of the cca index. We then test to see if the value we are about to test has previously been tested. If the value has been tested then we use the index of the first design that has not previously been tested. If the design has not previously been tested then we use that design. Lastly, we assign the selected value to the target design. After the new value has been assigned, we send the new version of our target design through metric testing to determine if the new value that was assigned to the target design has produced a valid design. We repeat this value selection process for replacing the value of the target variable using every other known successful value for the chosen critical variable. We repeat this process for every design within the tradespace of designs to acquire a percent success for permutation analysis.

Table 8

Permutation stability analysis source code

```
 # Attempt to randomly get the index of a value in the critical variable list
mechanicalModel = rowDict[mechModel]
chosenbscell = ""
chosenCCA = ""
numBscell = len(localVarValList[mechanicalModel])
chosenBscellIndex = random.randint(0, numBscell - 1)
curBscellIndex = 0
chosenCcaValIndex = 0
for bscell in localVarValList[chosenSeed]:
  if curBscellIndex == chosenBscellIndex:
      chosenbscell = bscell
      numCca = len(localVarValList[mechanicalModel][chosenbscell])
      chosenCcaIndex = random.randint(0, numCca - 1)
      curCcaIndex = 0
      for cca in localVarValList[mechanicalModel][chosenbscell]:
         if curCcaIndex == chosenCcaIndex:
            chosenCCA = cca
            numValForCca = 
len(localVarValList[mechanicalModel][chosenbscell][chosenCCA])
            chosenCcaValIndex = random.randint(0, numValForCca - 1)
            break
         else:
            curCcaIndex += 1
      break
  else:
      curBscellIndex += 1

# Once we have attempted to randomly choose a value to try for this row,
make sure we have a
# random index of a value that hasn't been tested so that we can meet our
percentage.
# If we don't get a random number that hasn't been tested, take the next
number that hasn't been tested
```

```
if
(localVarValList[mechanicalModel][chosenbscell][chosenCCA][chosenCc
aValIndex]['tested'] == True):
  seed = rowDict[CONST.seed]
  for bscell in localVarValList[rowDict[CONST.seed]]:
      for cca in localVarValList[seed][bscell]:
          for curCCAVal in range(0, len(localVarValList[seed][bscell][cca])):
              if (localVarValList[seed][bscell][cca][curCCAVal]['tested'] ==
False):
                  if firstIndexOfUntestedCcaVal == -1:
                      # store a reference into the structure to the first untested
value
                      firstIndexOfUntestedSeed = seed
                      firstIndexOfUntestedBscell = bscell
                      firstIndexOfUntestedCca = cca
                      firstIndexOfUntestedCcaVal = curCCAVal
                  numUntested += 1
              else:
                  numTested += 1

  randomPermute = \
  localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][f
irstIndexOfUntestedCca][
             firstIndexOfUntestedCcaVal]['value']
   # if we are allowing duplicate tests of the same value
 localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][fi
rstIndexOfUntestedCca][
      firstIndexOfUntestedCcaVal]['tested'] = True
else:
  randomPermute =
localVarValList[mechanicalModel][chosenbscell][chosenCCA][chosenCc
aValIndex]['value']
   # if we are allowing duplicate tests of the same value
  if allowDuplicatePermutation == False:
      localVarValList[mechanicalModel][chosenbscell][chosenCCA][chosen
CcaValIndex]['tested'] = True

# Assign the randomly or next chosen value to the critical var in prop dict
# rowDict contains a design and we are replacing the selected primary
variable vvalue with the permuted value for testing
rowDict[primaryVar] = randomPermute
```

In case the reader is curious why we bother with attempting to randomly choose a value from the list of possible values for the critical variable, we apply random selection because it is possible that the user may have a solution space that is too large to perform permutation testing for 100 percent of possible values for every design. The random selection is in place to provide a mechanism for performing permutation utilizing a percentage of possible available values for a design. As we wished to show the results of fully utilizing permutation analysis, we did not provide results of percentage of possible results but the option is there in case the reader finds themself in a position where their solution space is to large to perform full permutation testing. It is recommended that if the reader chooses to perform permutation testing on a percentage of the population, the reader should make a note of the percentage of possible values that were tested so that it is clear that the results of permutation analysis represent the success rate for the subset of possible values.

Step 2 Part 2: Permutation Stability Analysis - Calculating Robustness with Mutation

A Genetic Algorithm Substitution Method for Finding Resilient Designs within a Tradespace

In order to provide more functionality to permutation analysis, it was thought that providing a means for the designer to explore the design space around successful designs would be helpful. Mutation analysis was added to permutation analysis in order to provide a means for the designer to locate designs that were not previously considered. Because the selection of designs through mutation analysis is random, new values may or may not be feasible. However, mutation analysis still provides a more enhanced view than permutation analysis alone by adding to the expectation of a design to be able to withstand changes to design parameters.

Possible Choices. Mutation analysis is an enhancement to permutation analysis that performs a random selection of values utilizing a target design, and a selected design from the design tradespace of possible designs. Mutation randomly chooses to select a value in between, below, or above the two selected values.

In Between. For the in-between choice, mutation must decide if it wants to be closer to the current value or the target value. After choosing the current or target value, then mutation checks to see if its choice is above or below the halfway point between the current and target value and it uses that information for ensuring the randomly chosen values are closer to the selection of the current or target value. The in between choice also has the option of selecting halfway in between the current and target value but if mutation chooses halfway then it does not matter if the mutation selects the current or target value because halfway is the same answer for both options.

Below and Above. If mutation chooses below or above then it must select the target value or the current value. Once mutation chooses the target value or the current value, it randomly chooses a number in between the halfway point and its choice of the target or current value. Mutation then subtracts or adds the randomly chosen value with its selection between the current and target value based on if it wants the mutated value to be above or below the current or selected value.

A sudo code example of mutation analysis is now provided:

Table 9

Calculate mutated value

```
''' Mutation possible answers
  0) Somewhere in between
      a) Random value closer to current value
      b) Random value closer to target value
      c) Halfway
      Note: If the new value is already present, then move on without mutation
  1) Above or below current value by whichever puts the point:
      a) Places the current value in between itself and the target value ( new <---- current -------
target )
      b) Places the target value in between itself and the current value ( current ----- target ----->
new ) '''
def mutateValue(current, target):
  # choice 0 or 1
   # 0) Somewhere in between
   # 1) Above or below current value by whichever puts the point:
  position = [0, 1]
  positionChoice = random.choice(position)

  # in between choice
   # 0) Random value closer to current value
   # 1) Random value closer to target value
   # 2) Halfway
  inBetween = [0, 1, 2]
  inBetweenChoice = random.choice(inBetween)

  # aboveBelowChoice
   # 0) Places the current value in between itself and the target value ( new <---- current -------
target )
   # 1) Places the target value in between itself and the current value ( current ----- target ----->
new )
  aboveBelow = [0, 1]
  aboveBelowChoice = random.choice(aboveBelow)

  # only dealing with positive numbers
  mutatedValue = 0
  halfway = (current+target)/2

  current = int(current)
  halfway = int(halfway)
  target = int(target)

   # force a mutated range
  if halfway == target or halfway == current:
      current = random.randrange(600,1000)
      target = random.randrange(0,400)
      halfway = 500

  current = int(current)
  halfway = int(halfway)
  target = int(target)

   # Somewhere in between
```

```
 if positionChoice == 0:
     # Random value closer to current value
    if inBetweenChoice == 0:
        if(current < halfway):
            mutatedValue = random.randrange(current, halfway,1)
        else:
            mutatedValue = random.randrange(halfway, current,1)

     # Random value closer to target value
    elif inBetweenChoice == 1:
        if(target < halfway):
            mutatedValue = random.randrange(target, halfway,1)
        else:
            mutatedValue = random.randrange(halfway, target,1)

     # Halfway
    elif inBetweenChoice == 2:
        mutatedValue = halfway
 else: #Above or below current value by whichever puts the point
     # Places the current value in between itself and the target value ( new <---- current -------
target )
    if aboveBelowChoice == 0:
         # ( new <---- current ------- target )
        if(current < halfway):
            mutatedValue = current - random.randrange(current, halfway, 1)
        else: # ( target ----- current -------> new )
            mutatedValue = current + random.randrange(halfway, current, 1)

     # Places the target value in between itself and the target value ( current ----- target ----->
new )
    elif aboveBelowChoice == 1:
        if(target < halfway):
             # ( new <---- target ----- current )
            mutatedValue = target - random.randrange(target, halfway,1)
        else:
             # ( current ----- target -----> new )
            mutatedValue = target + random.randrange(halfway, target,1)

 return mutatedValue
```

CHAPTER IV – RESULTS

Introduction to Results and the Initial Statistics of the SSCTF Dataset

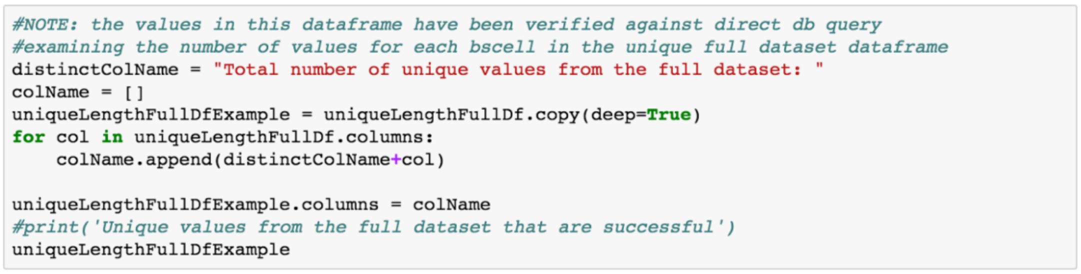
We begin processing Small Surface Combatant Task Force (SSCTF) data (described fully in the Introduction section) with basic statistics: Non-Unique designs, Unique designs, Successful Unique designs, Unsuccessful Unique designs, and Summary Statistics for each of the Five Mechanical Models. These statistical measures were developed during the SSCTF project and are presented here as the base methodology upon which we are improving. Code segments and derived tables and graphs are included for completeness.

Following basic statistics as developed during SSCTF, we introduce results from design permutation and the additional insight gained on the permutation stability of the four key characteristics of successful multi-purpose surface ships: Free Weight, Free Power, Free Cooling and Free Space. These four characteristics are described fully in the Introduction section. This extended methodology is applied to the problem of selecting a surface ship mechanical model (described in the Introduction) that is both likely to be a successful ship, meeting all key metrics (described in the methodology) and which is most likely to survive the uncertainty bid and manufacturing process, preserving the four key characteristics.

Code and derived tables and graphs are presented to support the utility of this new methodology and potential usage in current exploratory clean-sheet undersea design and other upcoming joint projects.

Non-Unique Designs

We begin by first looking at the total number of non-unique designs. All mechanical models possess the same number of non-unique designs. We theorize that this is because the data generated was focused on producing a uniform data set. It is convenient that the data is equally present for all mechanical models for the sake of a fair comparison of the results of permutation stability analysis.



```
#NOTE: the values in this dataframe have been verified against direct db query
#examining the number of values for each bscell in the unique full dataset dataframe
distinctColName = "Total number of unique values from the full dataset: "
colName = []
uniqueLengthFullDfExample = uniqueLengthFullDf.copy(deep=True)
for col in uniqueLengthFullDf.columns:
    colName.append(distinctColName+col)

uniqueLengthFullDfExample.columns = colName
#print('Unique values from the full dataset that are successful')
uniqueLengthFullDfExample
```

Figure 37. Total number of unique designs from the full dataset

Table 10

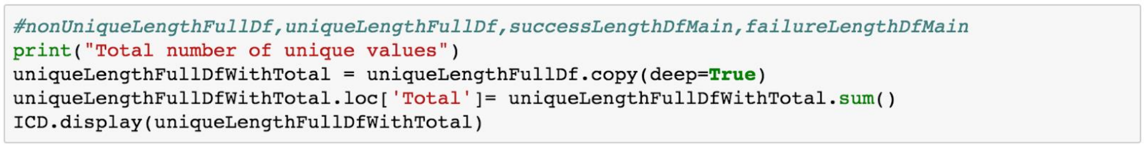
The full dataset with all non-unique designs

Bscell	**Total number of non-unique designs in the full dataset: i1**	**Total number of non-unique designs in the full dataset: i2**	**Total number of non-unique designs in the full dataset: ic**	**Total number of non-unique designs in the full dataset: m1**	**Total number of non-unique designs in the full dataset: m2**
1A	1900	1900	1900	1900	1900
1A-DF-1	1900	1900	1900	1900	1900
1A-D2-2	1900	1900	1900	1900	1900
2A	2100	2100	2100	2100	2100
2A-DF-1	2100	2100	2100	2100	2100
2A-D2-2	2100	2100	2100	2100	2100
3A	7800	7800	7800	7800	7800
3A-DF-1	7800	7800	7800	7800	7800

3A-D2-2	7800	7800	7800	7800	7800
4A	6550	6550	6550	6550	6550
4A-DF-1	6550	6550	6550	6550	6550
4A-D2-2	6550	6550	6550	6550	6550
5A	3400	3400	3400	3400	3400
5A-DF-1	3400	3400	3400	3400	3400
5A-D2-2	3400	3400	3400	3400	3400
6A	2700	2700	2700	2700	2700
6A-DF-1	2700	2700	2700	2700	2700
6A-D2-2	2700	2700	2700	2700	2700
7A	10200	10200	10200	10200	10200
7A-DF-1	10200	10200	10200	10200	10200
7A-D2-2	10200	10200	10200	10200	10200
8A	8450	8450	8450	8450	8450
8A-DF-1	8450	8450	8450	8450	8450
8A-D2-2	8400	8400	8400	8400	8400
Total	129250	129250	129250	129250	129250

Unique Designs

As expected, we can see in Table 11 below, that each Bscell has the same number of unique designs across all mechanical models. While Bscells may have different numbers of unique designs, all mechanical models possess the same total number of unique designs.



```
#nonUniqueLengthFullDf,uniqueLengthFullDf,successLengthDfMain,failureLengthDfMain
print("Total number of unique values")
uniqueLengthFullDfWithTotal = uniqueLengthFullDf.copy(deep=True)
uniqueLengthFullDfWithTotal.loc['Total']= uniqueLengthFullDfWithTotal.sum()
ICD.display(uniqueLengthFullDfWithTotal)
```

Figure 38. Total number of unique designs from the full dataset

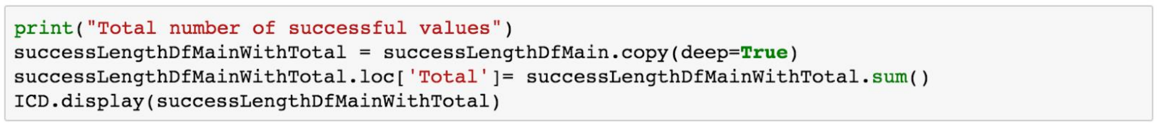
Table 11

Total number of unique designs from the full dataset

Bscell	**i1**	**i2**	**ic**	**m1**	**m2**
1A	38	38	38	38	38
1A-DF-1	38	38	38	38	38
1A-D2-2	38	38	38	38	38
2A	42	42	42	42	42
2A-DF-1	42	42	42	42	42
2A-D2-2	42	42	42	42	42
3A	156	156	156	156	156
3A-DF-1	156	156	156	156	156
3A-D2-2	156	156	156	156	156
4A	131	131	131	131	131
4A-DF-1	131	131	131	131	131
4A-D2-2	131	131	131	131	131
5A	68	68	68	68	68
5A-DF-1	68	68	68	68	68
5A-D2-2	68	68	68	68	68
6A	54	54	54	54	54
6A-DF-1	54	54	54	54	54
6A-D2-2	54	54	54	54	54
7A	204	204	204	204	204
7A-DF-1	204	204	204	204	204
7A-D2-2	204	204	204	204	204
8A	169	169	169	169	169
8A-DF-1	169	169	169	169	169
8A-D2-2	168	168	168	168	168
Total	2585	2585	2585	2585	2585

Successful Unique Designs for Each Mechanical Model

In table 12 we can see that not all bscells are equally successful before permutation. It can also be observed that for each bscell, we may have cases where a bscell is successful for some mechanical models, but not all mechanical models. So far in the observation of the data, this is the first point where we can see a difference in the performance of the different mechanical models. A successful design is one that passes all 16 performance metrics (explained in greater detail in the methodology). We also see variance in the number of successful designs among differing bscells. This can be explained by the complex interaction among differing capabilities concepts and mechanical models. For example, with some mechanical models, one bscell may have a capability that leads to a longer hull length which then leads to a larger slower ship and may not pass all metrics.



```
print("Total number of successful values")
successLengthDfMainWithTotal = successLengthDfMain.copy(deep=True)
successLengthDfMainWithTotal.loc['Total']= successLengthDfMainWithTotal.sum()
ICD.display(successLengthDfMainWithTotal)
```

Figure 39. Code used for producing the successful unique designs chart

Table 12

Successful unique designs for each mechanical model

Bscell	**i1**	**i2**	**ic**	**m1**	**m2**
1A	0	0	0	2	34
1A-DF-1	31	38	38	38	38
1A-D2-2	5	26	26	26	26
2A	0	0	1	7	41
2A-DF-1	41	42	42	42	42
2A-D2-2	29	42	42	42	42
3A	0	0	0	12	139
3A-DF-1	124	156	156	156	156
3A-D2-2	25	78	78	78	78
4A	0	0	0	5	97
4A-DF-1	85	131	131	131	131
4A-D2-2	13	81	83	83	83

5A	0	0	0	12	67
5A-DF-1	68	68	68	68	68
5A-D2-2	55	68	68	68	68
6A	0	0	0	9	50
6A-DF-1	53	54	54	54	54
6A-D2-2	28	47	47	47	47
7A	0	0	0	24	182
7A-DF-1	172	204	204	204	204
7A-D2-2	0	0	0	0	0
8A	0	0	0	13	127
8A-DF-1	113	169	169	169	169
8A-D2-2	0	0	0	0	0
Total	842	1204	1207	1290	1943

Number of Failures for Each Mechanical Model

The number of failures for each mechanical model could have been inferred from the total number of unique designs and the total number of designs that passed the 16 metrics, and also the requirements testing. However, it is helpful to have the total number of failures in tabular format. We close the description of the failures data by observing that some bscells have no failures implying that these bscells are likely to be pass all 16 metrics regardless of mechanical model.

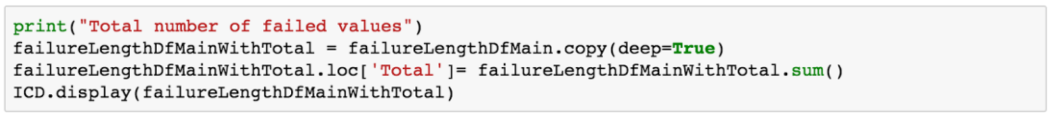

```
print("Total number of failed values")
failureLengthDfMainWithTotal = failureLengthDfMain.copy(deep=True)
failureLengthDfMainWithTotal.loc['Total']= failureLengthDfMainWithTotal.sum()
ICD.display(failureLengthDfMainWithTotal)
```

Figure 40. Code to pull of the number of failures for each mechanical model

Table 13

Number of failures for each mechanical model

Bscell	**i1**	**i2**	**ic**	**m1**	**m2**
1A	38	38	38	36	4
1A-DF-1	7	0	0	0	0
1A-D2-2	33	12	12	12	12
2A	42	42	41	35	1
2A-DF-1	1	0	0	0	0
2A-D2-2	13	0	0	0	0
3A	156	156	156	144	17
3A-DF-1	32	0	0	0	0
3A-D2-2	131	78	78	78	78
4A	131	131	131	126	34
4A-DF-1	46	0	0	0	0
4A-D2-2	118	50	48	48	48
5A	68	68	68	56	1
5A-DF-1	0	0	0	0	0
5A-D2-2	13	0	0	0	0
6A	54	54	54	45	4
6A-DF-1	1	0	0	0	0
6A-D2-2	26	7	7	7	7
7A	204	204	204	180	22
7A-DF-1	32	0	0	0	0
7A-D2-2	204	204	204	204	204
8A	169	169	169	156	42
8A-DF-1	56	0	0	0	0
8A-D2-2	168	168	168	168	168
Total	1743	1381	1378	1295	642

Combined Statistics for Each of the Mechanical Models

Lets condense the previous summary statistics data down into a single table (table 5) showing overall mechanical model performance. We can see that the mechanical model, M2, passes testing 75% of the time. Mechanical model I2, Ic, and M1 pass testing 46-49 percent of the time. Mechanical model I2 comes in last with the lowest pre-permutation testing with only 32% of unique designs passing testing. Thus, a design from the mechanical model family M2 is estimated to be 26 percent more likely than the alternative mechanical models to pass all required metrics following the bid and manufacture process. This is a significant difference and without any further analysis, M2 would be the best choice when seeking a design likely to be a successful ship "as built." We can see that in the data provided, M2 has nearly 2 times the number of rows of data that were present at the start before permutation. This is acceptable as we are focused on estimating the likelihood a design will still be successful following "intra-mechanical model" design parameter swaps.

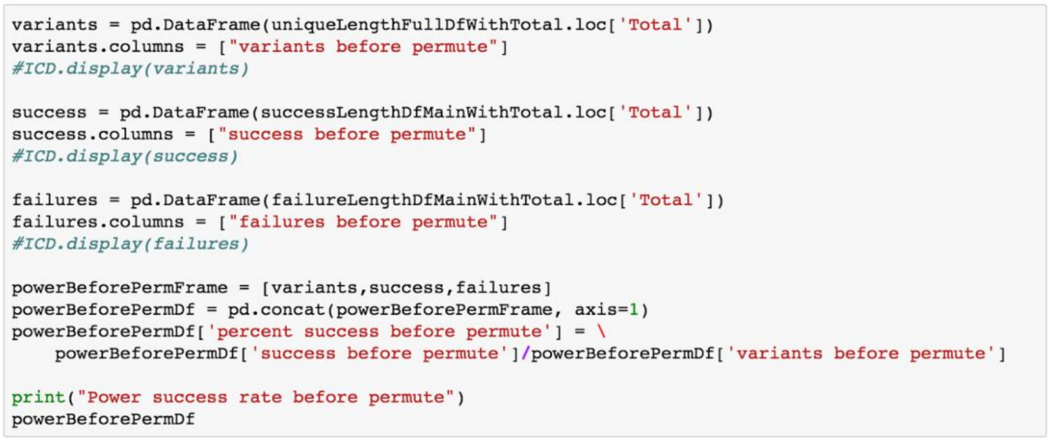
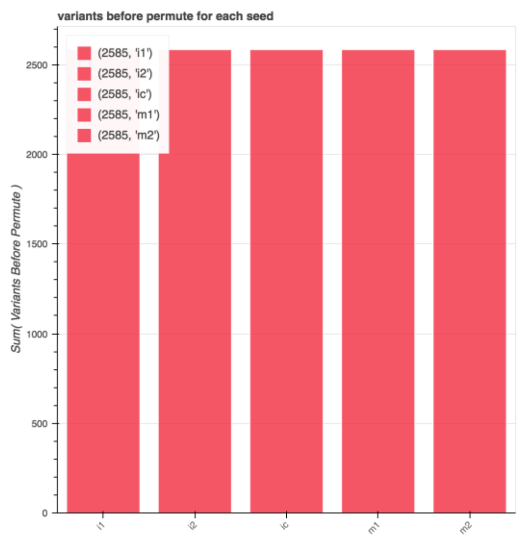

```
variants = pd.DataFrame(uniqueLengthFullDfWithTotal.loc['Total'])
variants.columns = ["variants before permute"]
#ICD.display(variants)

success = pd.DataFrame(successLengthDfMainWithTotal.loc['Total'])
success.columns = ["success before permute"]
#ICD.display(success)

failures = pd.DataFrame(failureLengthDfMainWithTotal.loc['Total'])
failures.columns = ["failures before permute"]
#ICD.display(failures)

powerBeforePermFrame = [variants,success,failures]
powerBeforePermDf = pd.concat(powerBeforePermFrame, axis=1)
powerBeforePermDf['percent success before permute'] = \
    powerBeforePermDf['success before permute']/powerBeforePermDf['variants before permute']

print("Power success rate before permute")
powerBeforePermDf
```

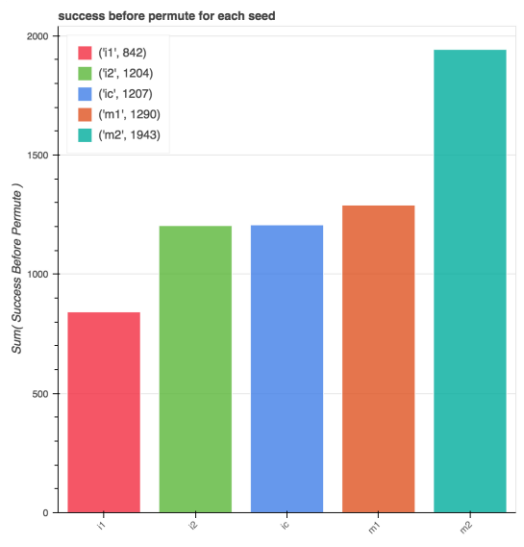
Figure 41. Combined statistics for each of the mechanical models

Table 14

Combined results showing pre-permutation statistics

Mechanical mode	variants before permute	success before permute	failures before permute	percent success before permute
i1	2585	842	1743	0.325725
i2	2585	1204	1381	0.465764
ic	2585	1207	1378	0.466925
m1	2585	1290	1295	0.499033
m2	2585	1943	642	0.751644





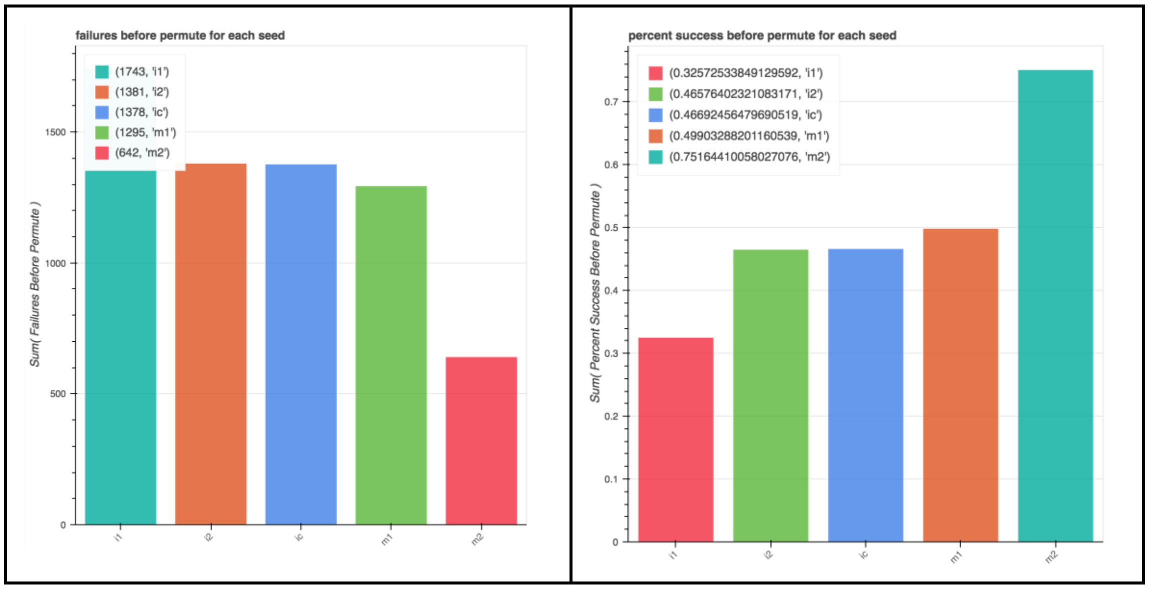


Figure 42. Pre-permutation statistics for mechanical models

Table 14 represents the current "state of art" in the SSCTF design selection process. In the next section, we will take a quick look at computation running times. After computation running times, the following section will begin *an enhancement and refinement to the current process of design selection* with an addition to the SSCTF design process. This enhancement to the design process will be known as *design permutation computations and analysis.*

Algorithmic and Performance Issues

Although the primary focus of this study is the development of a new analysis methodologies with existing SSCTF data which is the neighborhood of 27Mb, algorithmic and computational efficiency may be important in future studies in which potentially billions of designs are evaluated. Database reads were accomplished in 12.24 seconds. Parallel (using the MPI4Py library) permutation analysis on an 8 core (16 thread) Mac Pro is summarized in Table 6.

Table 15

Timings

Mechanical Model	Time (Seconds) config data	Number of Rows	Time (seconds) it took for permute	Avg Calculation Time (seconds) per row
m2	5.10	30978	10741	2.88
m1	2.29	16951	3481	4.86
ic	1.97	14332	2439	5.82
i1	.34	2305	227	10.15
i2	2.00	14353	2314	6.20

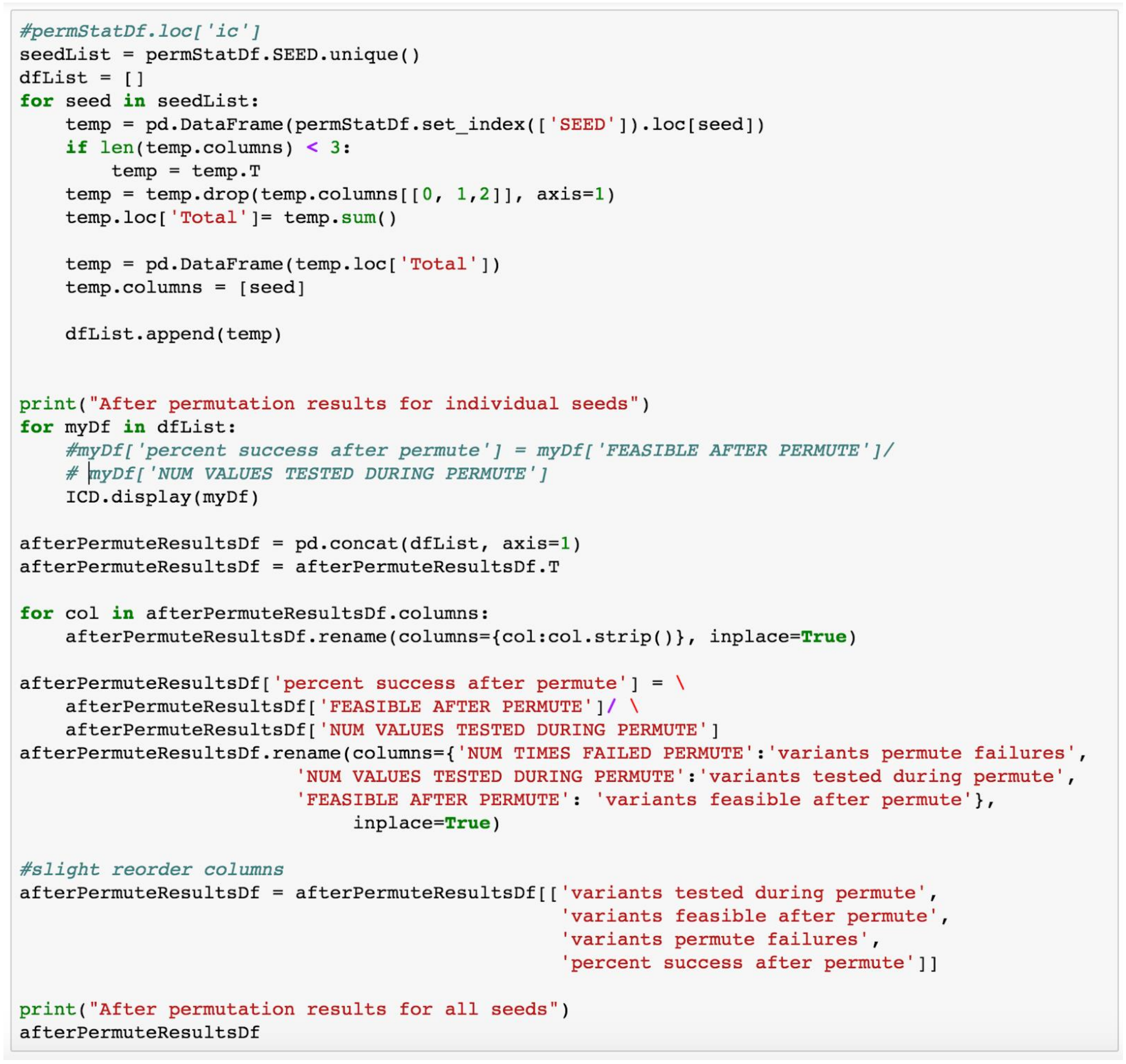
Time it took for Configuring the data structures, permuting the data, and the number of unique designs processed for each mechanical model

The permutation calculation average time per row seems to do better with more rows of data, but at best we can expect to spend ~2.8 seconds per row. The current implementation adapts to the number of cores and could easily scale to much larger problems. Eventually, this could be extended to HPC either using MPI4Py or the HPC Modernization Program Galaxy Orchestration platform.

Permutation

Introduction to Permutation

In this section, we will look at the results of permutation analysis. At this point the code used is the same for all mechanical models, however, the results of permutation on each of the 4 critical variables (Power, Space, Cooling, Weight) have different results. Let's first take a quick look at the code before we look at the results from permutation on the 4 critical variables for each of the 5 mechanical models. Although the code below prints out individual tables for each mechanical model, a combined table showing all mechanical models in one location will be presented for each of the critical variables in their respective sections. Finer details of the permutation code and algorithm are in the Methodology Section.



```
#permStatDf.loc['ic']
seedList = permStatDf.SEED.unique()
dfList = []
for seed in seedList:
    temp = pd.DataFrame(permStatDf.set_index(['SEED']).loc[seed])
    if len(temp.columns) < 3:
        temp = temp.T
    temp = temp.drop(temp.columns[[0, 1,2]], axis=1)
    temp.loc['Total']= temp.sum()

    temp = pd.DataFrame(temp.loc['Total'])
    temp.columns = [seed]

    dfList.append(temp)


print("After permutation results for individual seeds")
for myDf in dfList:
    #myDf['percent success after permute'] = myDf['FEASIBLE AFTER PERMUTE']/
    # myDf['NUM VALUES TESTED DURING PERMUTE']
    ICD.display(myDf)

afterPermuteResultsDf = pd.concat(dfList, axis=1)
afterPermuteResultsDf = afterPermuteResultsDf.T

for col in afterPermuteResultsDf.columns:
    afterPermuteResultsDf.rename(columns={col:col.strip()}, inplace=True)

afterPermuteResultsDf['percent success after permute'] = \
    afterPermuteResultsDf['FEASIBLE AFTER PERMUTE']/ \
    afterPermuteResultsDf['NUM VALUES TESTED DURING PERMUTE']
afterPermuteResultsDf.rename(columns={'NUM TIMES FAILED PERMUTE':'variants permute failures',
                        'NUM VALUES TESTED DURING PERMUTE':'variants tested during permute',
                        'FEASIBLE AFTER PERMUTE': 'variants feasible after permute'},
                             inplace=True)

#slight reorder columns
afterPermuteResultsDf = afterPermuteResultsDf[['variants tested during permute',
                                               'variants feasible after permute',
                                               'variants permute failures',
                                               'percent success after permute']]

print("After permutation results for all seeds")
afterPermuteResultsDf
```

Figure 43. Produce charts for all mechanical models and a combine chart of results

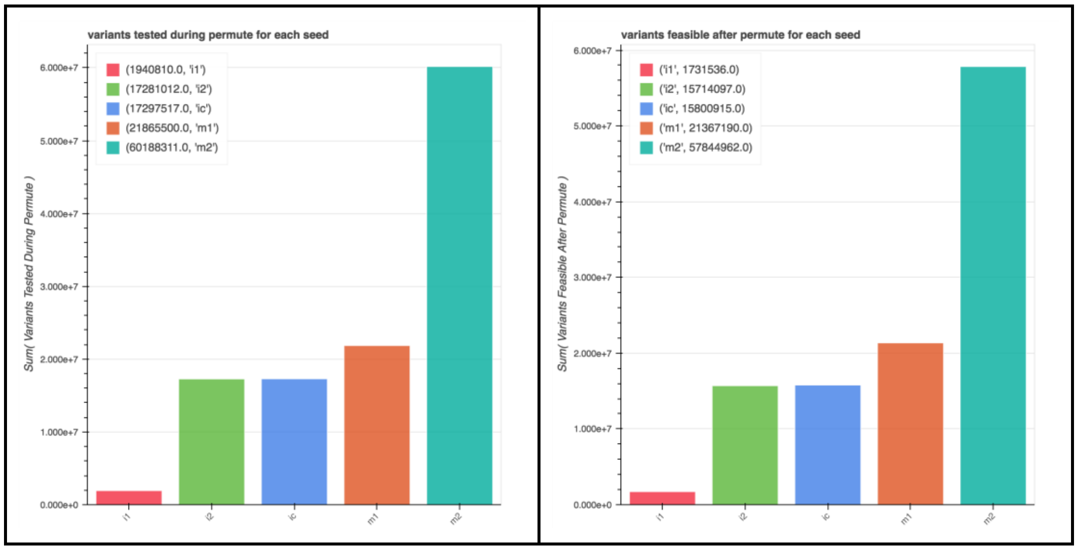
Space Permutation Results

Space is the first critical variable on which we observe the results of permutation. We can immediately notice that for the mechanical models, M1 is slightly better than M2 at handling permutation on the critical variable *Space*. We look back to the summary statistic for the mechanical models and note that before permutation, M2 was 75 percent likely to pass the 16 testing metrics, when M1 was 49 percent likely to pass those same metrics. This means that if we have a passing design for both M1 and M2, and we care most about whether or not the design will be able to handle changes in *Space*, we should choose M1 over M2. At this point, *we can see that permutation has already shown that it has the potential to influence the mechanical model selection process.*

Table 16

After permutation results for all mechanical models on critical variable Space

Mechanical Model	variants tested during permute	variants feasible after permute	variants permute failures	percent success after permute
i1	1940810.0	1731536.0	209274.0	0.892172
i2	17281012.0	15714097.0	1566915.0	0.909327
ic	17297517.0	15800915.0	1496602.0	0.913479
m1	21865500.0	21367190.0	498310.0	0.977210
m2	60188311.0	57844962.0	2343349.0	0.961066



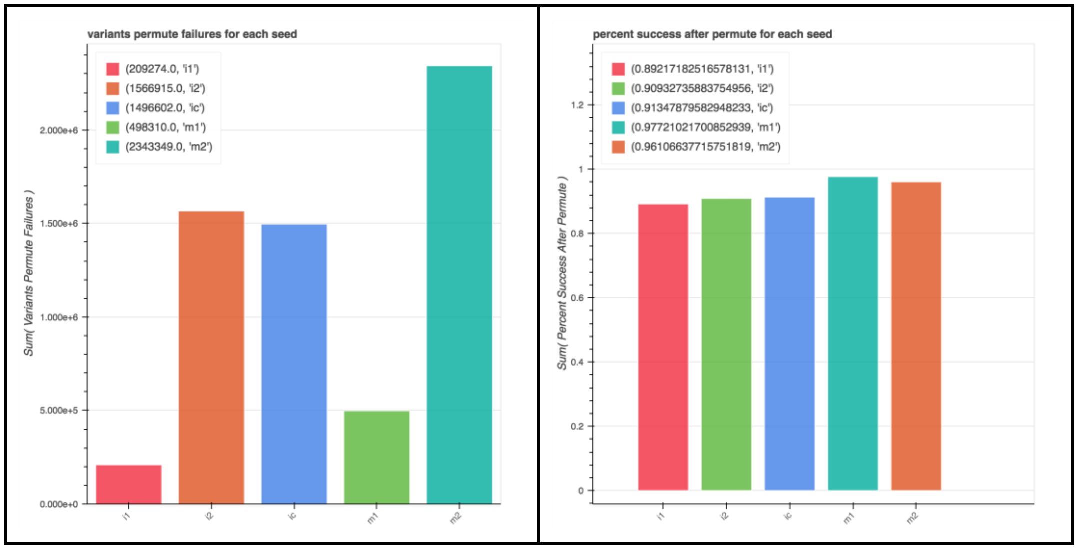


Figure 44. Space - Combined bar chart statistics

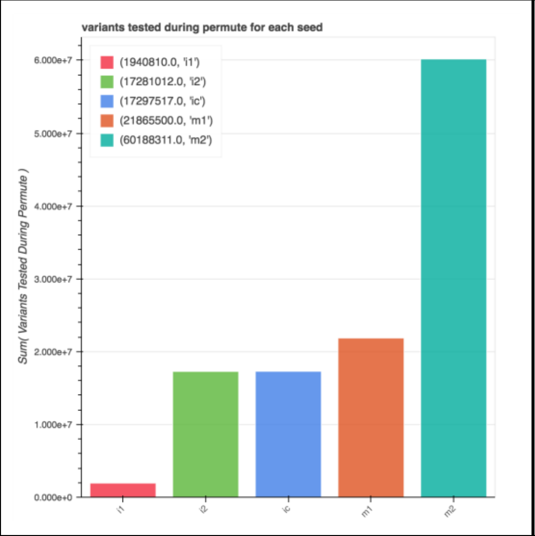
Weight Permutation Results

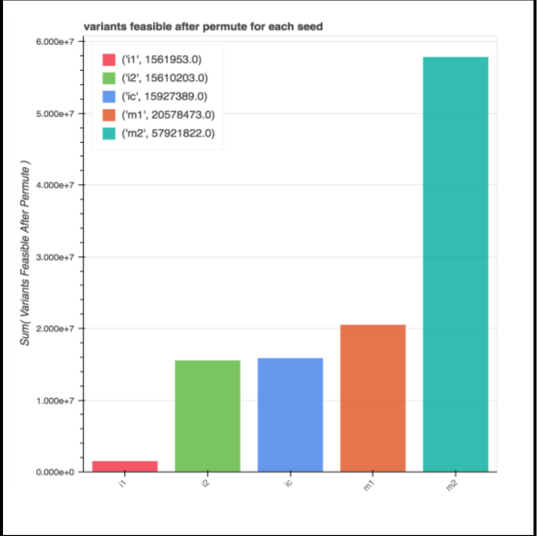
After examination of the *Space* critical variable, and noting that mechanical model M1 is the optimal choice when looking at a design's ability to withstand changes to the *Space* variable, we move on to examine the results of permutation on the critical variable *Weight*. Looking at the *Weight* critical variable, we can see that M2 is ~2 percent better at handling permutation to the *Weight* critical variable than M1. It is surprising that M1 did not perform as well as M2 when handling permutation to the critical variable Weight because M2 has two propellers and would assumingly possess less free weight to use than M1, which only has one propeller. Because there is a complex relationship between engine performance and fuel consumption to obtain the required ship range, it is possible that having two screws (M2) is more efficient than one screw (M1) therefore requiring less fuel and more favorable weight permutation stability. However, this reasoning for M2 having a better robustness score for the critical variable weight is speculative. If the designer cares mostly about a design's ability to handle changes as observed through permutation to the critical variable Weight, then the designer should choose mechanical model M2. Again, the additional layer of analysis produces interesting and potentially important information.

Table 17

After permutation results for all mechanical models on critical variable Weight

Mechanical Model	variants tested during permute	variants feasible after permute	variants permute failures	percent success after permute
i1	1940810.0	1561953.0	378857.0	0.804794
i2	17281012.0	15610203.0	1670809.0	0.903315
ic	17297517.0	15927389.0	1370128.0	0.920790
m1	21865500.0	20578473.0	1287027.0	0.941139
m2	60188311.0	57921822.0	2266489.0	0.962343





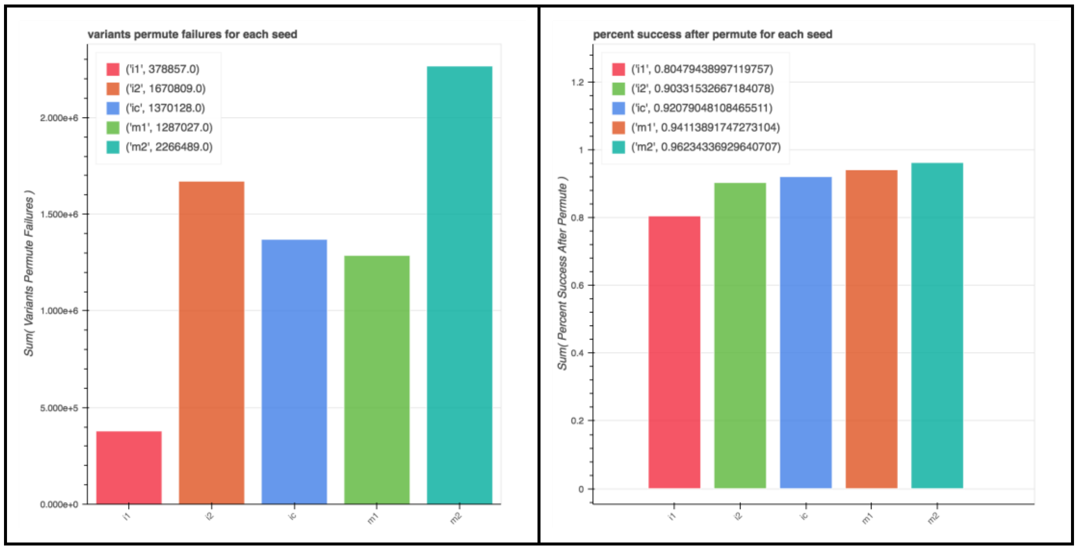


Figure 45. After permutation results for all seeds Weight

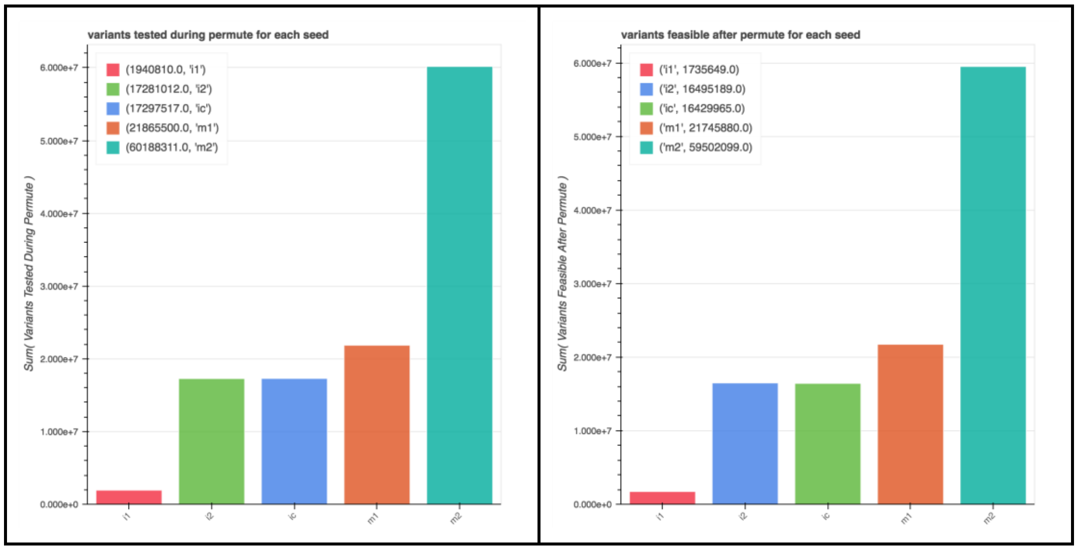
Power Permutation Results

So far we have determined that M1 is better at handling permutation on the critical variable *Space*, and M2 is better at handling permutation on the critical variable *Weight*. We now move on to examine the results of permutation on the critical variable *Power*. For all mechanical models, the results show that they are all very capable of handling changes to the critical variable *Power*. However, M1 is impressively able to handle changes to the critical variable *Power* 99 percent of the time. M1's closest competitor is M2 with 98 percent chance of handling changes to the critical variable *Power*. Even though 98 percent chance that M2 will handle a change to the critical variable power is really good, it is still slightly better for a designer interested mainly in a mechanical model's ability to withstand changes to power requirements to choose the mechanical model M1. Ship power is generated by bypassing an engine's main drive and diverting mechanical power to a generator. The current result is not a significant differentiating factor but could become one with finer detailed simulation of ship subsystems. As a side note, one of the primary attractions of electrical non-mechanical energy transfer systems such as *i1*, *i2* and mixed mode engines such as *ic* is survivability by avoiding situations where damage to a ship's main drive shuts down secondary power generation making a ship inoperable.

Table 18

After permeation results for all mechanical models on critical variable Power

Mechanical Model	variants tested during permute	variants feasible after permute	variants permute failures	percent success after permute
i1	1940810.0	1735649.0	205161.0	0.894291
i2	17281012.0	16495189.0	785823.0	0.954527
ic	17297517.0	16429965.0	867552.0	0.949845
m1	21865500.0	21745880.0	119620.0	0.994529
m2	60188311.0	59502099.0	686212.0	0.988599



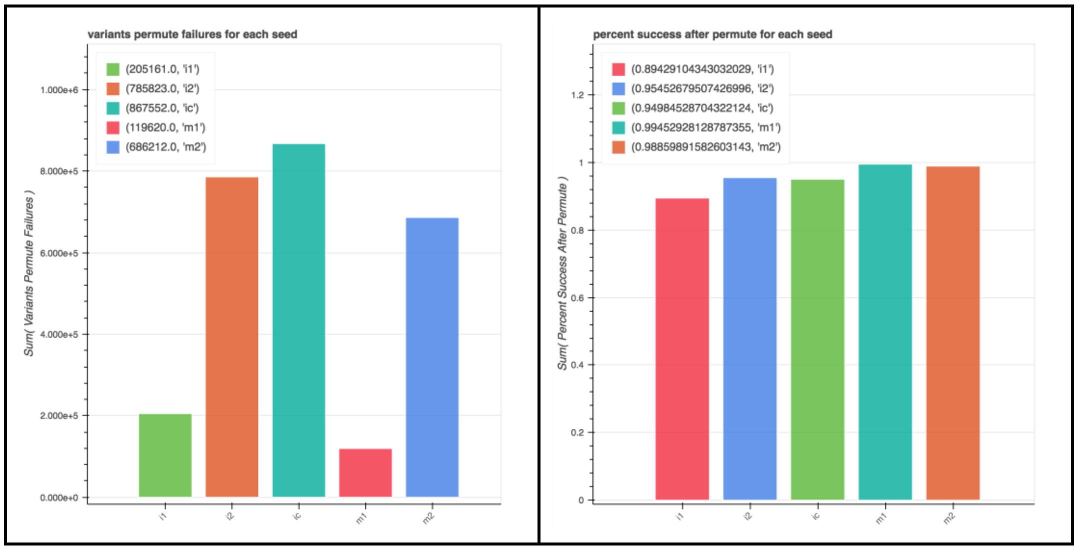


Figure 46. After permutation bar chart results for all mechanical models on critical variable Power

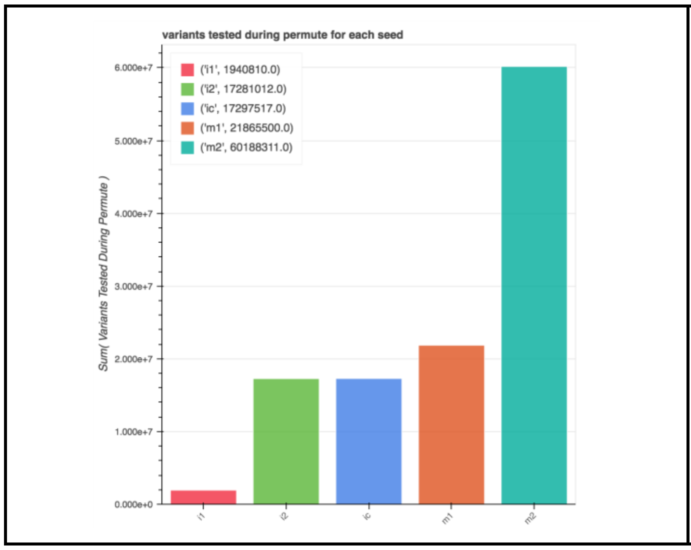
Cooling Permutation Results

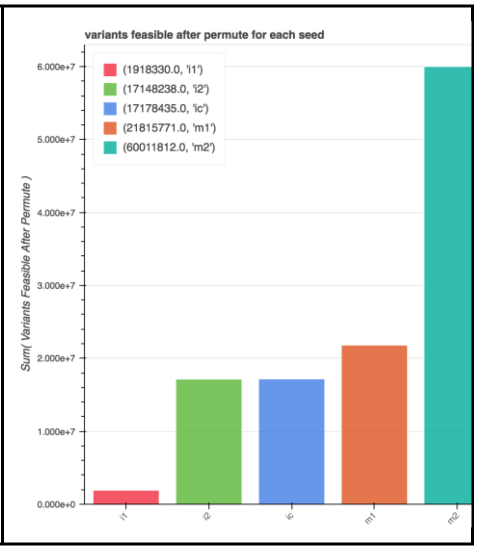
The last critical variable that we are going to perform permutation analysis on is *Cooling*. We have determined that mechanical model M1 is best at handling permutation on *Space* and *Power*, M2 is the optimal choice for handling permutation on *Weight*. Both M1 and M2 possess the ability to withstand permutation on the critical variable Cooling 97.7 percent of the time. This similar resistance to permutation on Cooling makes both M1 and M2 comparable choices when a designer cares mostly about a design's ability to withstand changes to the critical variable *Cooling*. As with ships free Power (above) Cooling was identified as a critical feature but was not modeling in sufficient detail to make useful inferences from permutation testing.

Table 19

After permutation results for all mechanical models on critical variable Cooling

Mechanical Model	variants tested during permute	variants feasible after permute	variants permute failures	percent success after permute
i1	1940810.0	1918330.0	22480.0	0.988417
i2	17281012.0	17148238.0	132774.0	0.992317
ic	17297517.0	17178435.0	119082.0	0.993116
m1	21865500.0	21815771.0	49729.0	0.997726
m2	60188311.0	60011812.0	176499.0	0.997068





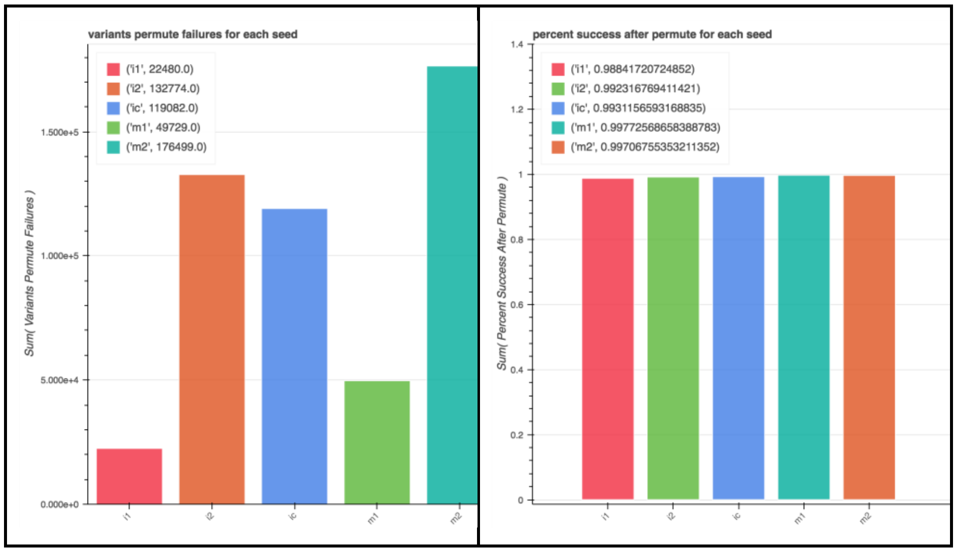
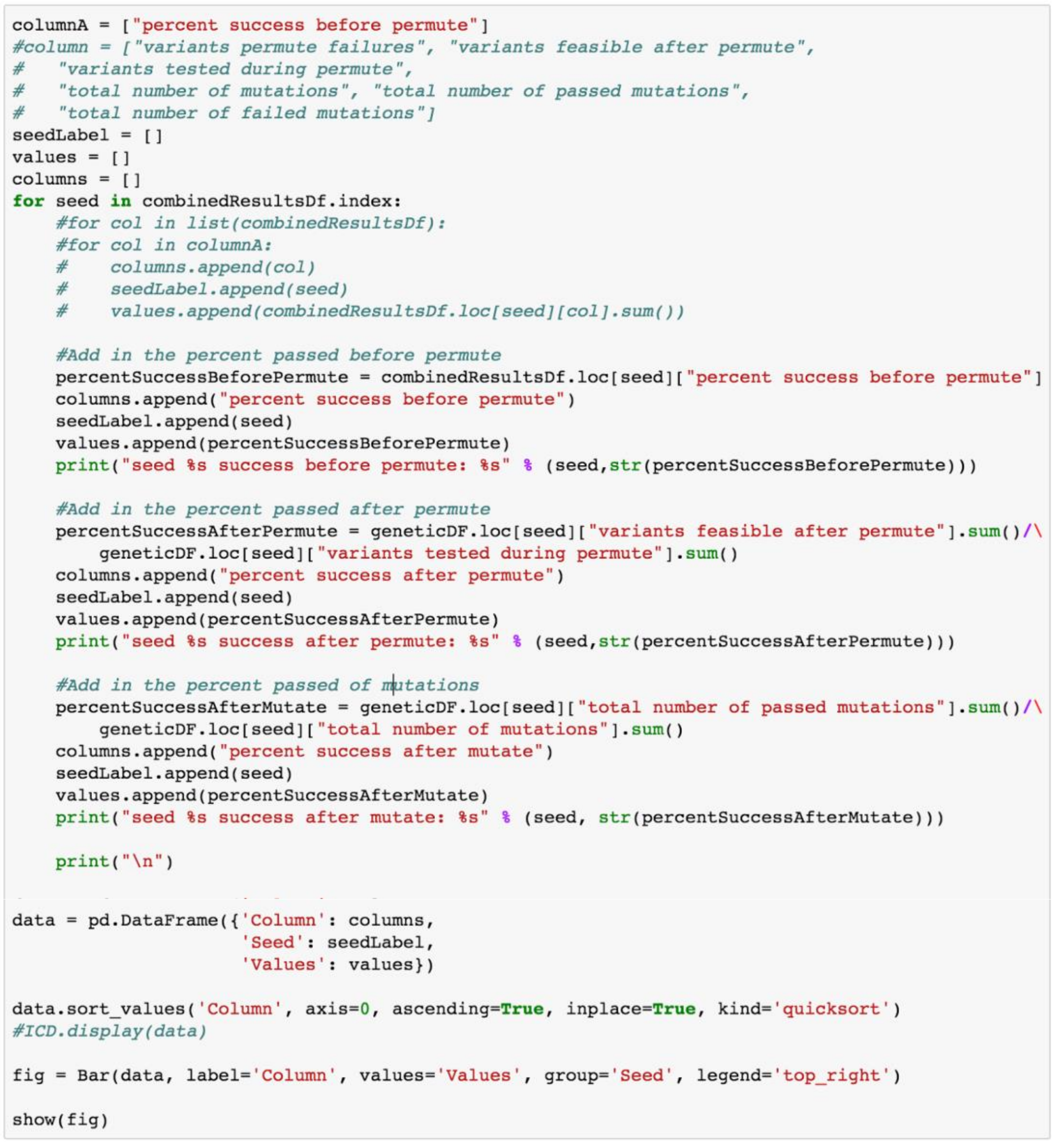


Figure 47. After permutation bar chart results for all mechanical models on critical variable Cooling

Mutation Analysis

Introduction to Mutation Analysis

Adding in *mutation analysis* (described in the Methodology) to the permutation process provided some results that may have revealed some bias in the data sets showing more focus on fully populating the range of possible values for M1 and M2. In order to help identify whether or not there was bias in the data, we present the results of Algorithm 2 (described in the Methodology). Algorithm 2 is the addition of mutation into the permutation stability analysis. Adding mutation to the permutation stability analysis adds additional knowledge in the form of helping to identify potential designs that may not have been considered or revealing if some mechanical models have had their solution space more fully explored than other mechanical models.



```
columnA = ["percent success before permute"]
#column = ["variants permute failures", "variants feasible after permute",
#    "variants tested during permute",
#    "total number of mutations", "total number of passed mutations",
#    "total number of failed mutations"]
seedLabel = []
values = []
columns = []
for seed in combinedResultsDf.index:
    #for col in list(combinedResultsDf):
    #for col in columnA:
    #    columns.append(col)
    #    seedLabel.append(seed)
    #    values.append(combinedResultsDf.loc[seed][col].sum())

    #Add in the percent passed before permute
    percentSuccessBeforePermute = combinedResultsDf.loc[seed]["percent success before permute"]
    columns.append("percent success before permute")
    seedLabel.append(seed)
    values.append(percentSuccessBeforePermute)
    print("seed %s success before permute: %s" % (seed,str(percentSuccessBeforePermute)))

    #Add in the percent passed after permute
    percentSuccessAfterPermute = geneticDF.loc[seed]["variants feasible after permute"].sum()/\
        geneticDF.loc[seed]["variants tested during permute"].sum()
    columns.append("percent success after permute")
    seedLabel.append(seed)
    values.append(percentSuccessAfterPermute)
    print("seed %s success after permute: %s" % (seed,str(percentSuccessAfterPermute)))

    #Add in the percent passed of mutations
    percentSuccessAfterMutate = geneticDF.loc[seed]["total number of passed mutations"].sum()/\
        geneticDF.loc[seed]["total number of mutations"].sum()
    columns.append("percent success after mutate")
    seedLabel.append(seed)
    values.append(percentSuccessAfterMutate)
    print("seed %s success after mutate: %s" % (seed, str(percentSuccessAfterMutate)))

    print("\n")

data = pd.DataFrame({'Column': columns,
                     'Seed': seedLabel,
                     'Values': values})

data.sort_values('Column', axis=0, ascending=True, inplace=True, kind='quicksort')
#ICD.display(data)

fig = Bar(data, label='Column', values='Values', group='Seed', legend='top_right')

show(fig)
```

Figure 48. Create and populate bar charts for mutation data

Space - Mutation Analysis

We begin looking at the ability of the 5 mechanical models to withstand mutation analysis to the 4 critical variables with examining the Space critical variable. We can see that the ability of all mechanical models to withstand mutation is a lower percentage than each of the mechanical models ability to withstand permutation. Having a lower percentage chance to withstand mutation than permutation is not surprising as mutation is exploring potential designs above, below, and in between selected designs. Despite the lower percentage chance to survive mutation, we can see that M1 is more likely to survive mutation than the alternative mechanical models when focusing on the *Space* critical variable in table 15 below.

Table 20

Effects of mutation on Space

Mechanical Model	success before permute	success after permute	success after mutate
i1	0.325725338491	0.8921718251657813	0.822094540794937
i2	0.465764023211	0.9093273588375496	0.8509669505471895
ic	0.466924564797	0.9134787958294823	0.856502240596922
m1	0.499032882012	0.9772102170085294	0.9608808347260953
m2	0.75164410058	0.9610663771575182	0.9330797611635518

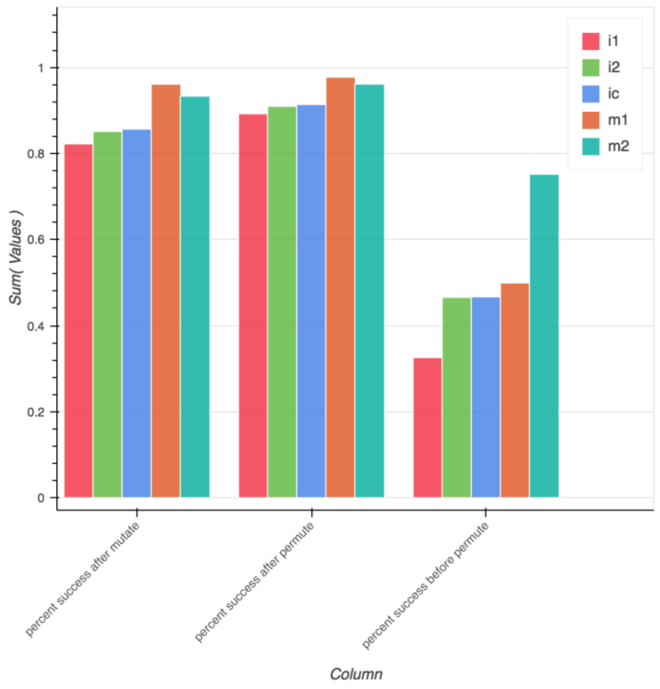


Figure 49. Bar chart comparing percent success after mutate and before permutation for *Space*

Weight - Mutation Analysis

When looking at the effects of mutation analysis on the space critical variable, the mechanical model M1 has the highest chance of producing a successful design. However, mechanical model M2 is the most capable of handling mutation of designs for the *Weight* critical variable. When looking at the ability of M2 to withstand changes to the weight critical variable, M2 is the most likely of the mechanical models to survive both mutation and permutation.

Table 21

Effects of mutation on Weight

Mechanical Model	success before permute	success after permute	success after mutate
i1	0.325725338491	0.8047943899711976	0.7294705350280938
i2	0.465764023211	0.9033153266718408	0.855368557527916
ic	0.466924564797	0.9207904810846551	0.8805951373521519
m1	0.499032882012	0.941138917472731	0.9097935947616783
m2	0.75164410058	0.9623433692964071	0.9402608557844796

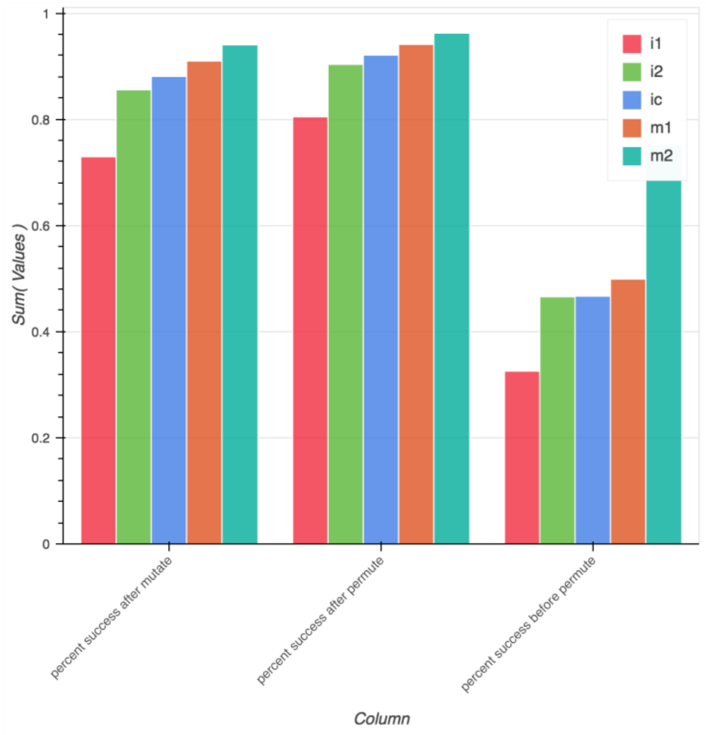


Figure 50. Bar chart comparing percent success after mutate and before permutation for Weight

Power - Mutation Analysis

The mechanical model M1 is better than M2 at handling mutation on the space critical variable, and M2 is better than M1 at handling mutation on the weight critical variable. But when it comes to handling the effects of mutation on the critical variable Power, M1 and M2 both seem to be only mildly affected by mutation. However, as we can see in Table 17, M1 is slightly better than M2 when mutation analysis is performed on the critical variable Power.

Table 22

Effects of mutation on Power

Mechanical Model	success before permute	success after permute	success after mutate
i1	0.325725338491	0.8942910434303203	0.866667559222817
i2	0.465764023211	0.95452679507427	0.9400133474972053
ic	0.466924564797	0.9498452870432212	0.9347440851921458
m1	0.499032882012	0.9945292812878735	0.9926935427987083
m2	0.75164410058	0.9885989158260314	0.9849463888436505

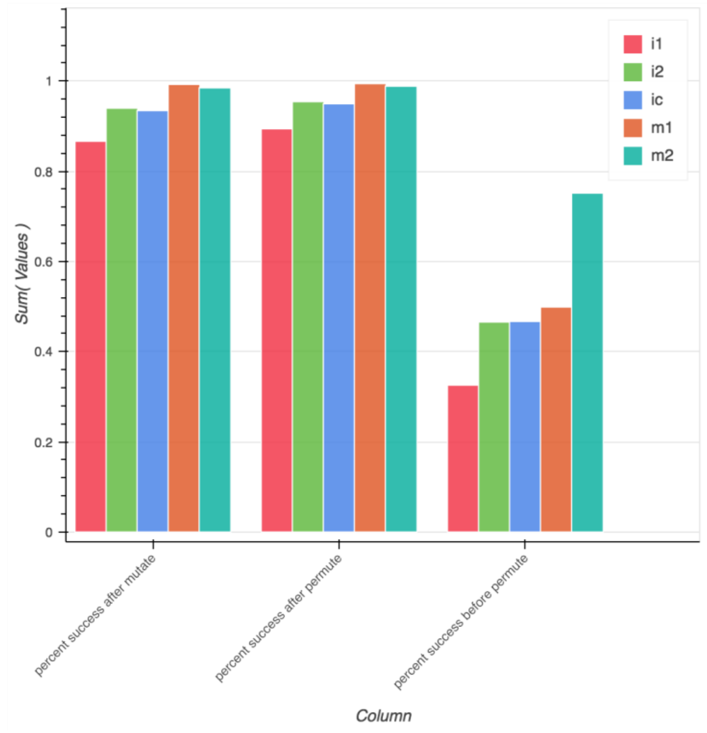


Figure 51. Bar chart comparing percent success after mutate and before permutation for Power

Cooling - Mutation Analysis

So far we have observed that M1 handles mutation analysis on both critical variables *Space and Power* slightly better than the alternative mechanical models, and M2 handles mutation to the *Weight* critical variable more efficiently than the other mechanical models. This leads us to our final critical variable that we performed mutation analysis upon, *Cooling*. We can see from the table below that all critical variables handle mutation on Cooling approximately the same with M1 and M2 performing slightly better. It was known that the critical variable *Cooling* had less data available to work with which may have led to the results of mutation on *Cooling* not having much effect on the results of mutation analysis.

Table 23

Effects of mutation on Cooling

Mechanical Model	success before permute	success after permute	success after mutate
i1	0.325725338491	0.98841720724852	0.9820829048571408
i2	0.465764023211	0.992316769411421	0.9880163887940463
ic	0.466924564797	0.9931156593168835	0.9890093823388916
m1	0.499032882012	0.9977256865838878	0.9965569819802315
m2	0.75164410058	0.9970675535321135	0.9955903820177083

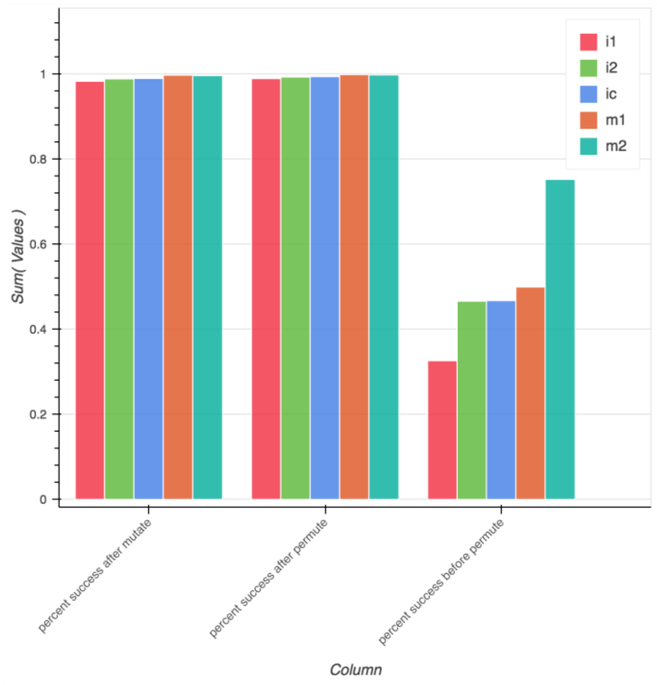


Figure 52. Bar chart comparing percent success after mutate and before permutation for Cooling

Mutation Conclusion

The results of mutation show that mechanical models M1 and M2 both handle mutation well. The results also show that the critical variable, *Space,* and *Weight* are more affected by mutation than the critical variables *Power* and *Cooling*. We can also note from the results of mutation that the data sets were fairly well populated with designs. Even though the effects of mutation were not that strong in the SSCTF dataset, mutation may have more impact in another dataset. Overall mutation analysis clarified Space and Weight permutation differences.

CHAPTER V – CONCLUSION

The objective of Robust Design is to model and make design choices which minimize risk of poor outcomes. By studying the effects of variants to design parameters, a designer can hope to soften the possible negative effects of changes to design parameters. Locating a design that is able to handle changes that negatively impact performance to design parameters throughout its lifecycle is known as a robust design. In many cases, the designer would prefer a robust design over an optimal design because the optimal design may not be able to handle changes to design parameters. This inability to handle changes to design parameters can lead to the optimal design no longer being a feasible design after even the smallest change to a design's parameters.

Summary of Objective

The study performed by the SSCTF was awarded the honor of being the best frigate based study the Navy had ever performed. Nevertheless, the work presented here was aimed at providing additional enhancements to this state of the art study. In order to facilitate this enhancement, this work sought to provide a methodology for design area experts to leverage when searching for a design capable of withstanding changes to design parameters in situations, such as the motivating SSCTF problem, where limited information is available on the likelihood of post-bid design changes. The method presented here was targeted, but not limited to, designers that do not have the luxury of a vertically integrated design process. The methodology takes into consideration that a designer may not have exact knowledge of the result of manufacturing which could have variation as a result of differences in the manufacturing process or if manufacturing uses exploratory means to find a way to fix unworkable design elements. This work utilized a data set provided by the Navy's Small Surface Combatant Task Force, however, the methodology does not require this data. This methodology can be applied to any dataset in which a designer faces the same or similar design complications as the Navy design teams face with manufacturing variance and a lack of vertical introspection.

Summary of the Methodology

The methodology represents two additional types of analysis beyond modeling done during the SSCTF. The first algorithm presented is a method to estimate design robustness utilizing the assembled population of ship designs as an estimate of possible as-manufactured variance. Four critical ship properties: Free Space, Free Weight, Free Power and Free Cooling are introduced and used in Algorithm 1 during permutation stability analysis on the design set. Permutation stability analysis is the process of utilizing known acceptable parameter values within other similar designs. By utilizing known acceptable values within similar designs helps to increase the likelihood that the borrowed parameter value will be an acceptable parameter value for the target design in the situation that no other constraints are know. The second algorithm of this methodology explores further by utilization of a genetic algorithm concept of mutation. The second algorithm added mutation by randomly choosing a value between, above, or below target design and the selected design. While some designs generated through mutation may have been infeasible without the need to test the value, mutating values in this way generated mostly acceptable results and has the potential to locate unidentified designs within explored areas of the tradespace. In addition, mutation also helps to simulate manufacturing variance and exploratory redefinition of design.

Concluding Results

We began the analysis of the Small Surface Combatant Task Force (SSCTF) dataset by populating the basic statistics: non-unique designs, unique designs, successful unique designs, unsuccessful unique designs and summary statistics for each of the five mechanical models. The basic statistics were presented as a summary of the base analysis methodology, which we are refining and extending.

For sake of completeness, we will briefly step through each of the basic statistics and provide a summary of their individual impact. We start with non-unique designs which allowed us to compare each of the mechanical models to determine if the mechanical models were equally populated within the dataset. Being equally present meant that the comparison of the results of permutation stability analysis would be a fair comparison. Next, we examined the number of unique designs. Because permutation utilized known alternative possible values when trading values between critical variables, it was more beneficial for permutation to trade unique values because trading non-unique values would produce a redundant result. For clarity, if the same design value is tested, it will produce the same result regardless of how many times it is tested. After identifying the number of unique designs for the full dataset, we looked at the number of successful unique designs for each mechanical model. By knowing the initial number of unique designs for each mechanical model we were able to identify that not all mechanical models possess the same number of successful designs. Not passing the same number of successful designs means that some mechanical models are more successful than others. Next, we produced a table showing the number of failures for each mechanical model. While the number of failures could have been derived from the number of unique values minus the number of successful unique values, it is helpful to have basic statistics easily viewable and accessible when performing data analysis. By having the number of unique failures easily viewable, we are able to view which mechanical models have the largest number of failed designs before permutation begins. We finish presenting the basic statistics for the mechanical models with a table which contained each of the previously mentioned statistics. We also included bar charts which provided a means for visible comparison of each of the basic statistics. By using the summary statistics table and bar charts, we were able to see that mechanical model M2 passed initial testing before permutation 75% of the time. Mechanical models I2, Ic, and M1 passed testing 26-29% less often than mechanical model M2. Lastly, I2 passed testing 32% of the time. Without any further analysis, M2 would be considered the optimal choice when seeking the best successful “as built” ship design. Utilizing the summary statistics *represented the current state of the art in the SSCTF design process.*

After finishing with the initial statistics, we then presented a short summary of algorithmic and performance issues. The SSCTF dataset was roughly 27MB. In the future, datasets may be larger increasing the need for algorithmic efficiency, however, the current parallel implementation was sufficient for the current dataset with total analysis runtime around 32 hours on an 8 core CPU. We presented a table of computation statistics with most notable statistic of permutation requiring a minimum of ~2.8 seconds per row.

After examining algorithmic and performance issues, we then looked at the results of our proposed *enhancement and refinement to the current design selection process* with *design permutation computation and analysis*. This proposed enhancement is also known as Algorithm 1. The results of permutation stability analysis were divided up into sections based on Free Space, Free Weight, Free Power, Free Cooling. These four ship characteristics were most important for SSCTF decision makers. Examples exist of ships that lack one or more of these characteristics and therefore have difficulty fulfilling their primary missions and are poor targets for modification to meet new and evolving future missions. Thus, designing for as much of the four "free" capability characteristics as possible was a major decision point and is a focus of this work. *We wish to provide additional analysis beyond SSCTF methodology to try to select designs most likely to have acceptable performance even with limited manufacturing information.*

For the SSCTF study performance was defined as passing 16 performance metrics such as range, speed, efficiency, and so forth. These metrics represent the capability to perform key missions for the designed ship. In manufacture Free Space, Free Weight, Free Power, Free Cooling could be consumed with unanticipated configuration changes leaving a ship with unsatisfactory performance. *The first analysis method beyond SSCTF methodology is "permutation" essentially taking the population of ships with similar mechanical design as a survey of the population of possible in-manufacture outcomes and swapping out key characteristics and determining if a ship still passed all 16 metrics.* This refinement is an estimate of the robustness of a design. Is the design simply a "sharp point" where performance degrades very badly with minor change orders or is the design in the center of a "plateau" of other good designs all of which will perform similarly well?

The first critical variable we examined for the results of permutation was Free Space. It was easily noticeable that M1 was better than its closest competitor, M2, at handling permutation on the critical variable *Space*. After looking back to the summary statistics for the mechanical models, we note that M2 was 75% likely to pass the 16 testing metrics, and M1 was 49% likely to pass those same testing metrics. The results of permutation on the critical variable space show that if we have a passing design for both of the critical variables, M1 and M2, and we care more about the design's ability to withstand changes in *Space* throughout its life cycle, we should choose M1 over M2. *We can see at this point that permutation has already shown that it has the potential to influence the mechanical model selection process.*

Next, we looked at the results of permutation on the critical variable *Weight*. We were able to determine that M2 is ~2 percent better at handling permutation on *Weight* than its closest competitor, M1. M2 possesses two mechanical screws and M1 possesses one mechanical screw making it surprising that M2 is better at handling changes to the *Weight* than M1. It is possible that M2 is better at handling permutation on the critical variable *Weight* than M2 because of the complex relationship between fuel efficiency and fuel weight. We observed that a designer caring most about a design's ability to handle changes to the critical variable *Weight* should choose the mechanical model M2. As with the critical variable Space, the results of permutation on the critical variable *Weight* produces additional interesting results.

Next, we continued on with observing the results of permutation on the critical variable *Power*. All of the mechanical models handled permutation on *Power* very well except for I1 with the lowest permutation score of 89%. The two top mechanical models for handling permutation to *Power* was M1 and M2 which were separated by only 1% stability, and the better of the two, M1, handling permutation 99% of the time. The difference in permutation stability on *Power* between M1 and M2 is very slight, however, it may be possible to enhance this difference by performing permutation stability on subsystems which contribute to the permutation success percentage of the critical variable *Power.*

We conclude the results of permutation on the 4 critical variables with the critical variable *Cooling*. M1 and M2 perform approximately the same from permutation on the critical variable *Cooling* with 97.7% success rate. In the SSCTF study, *Cooling* was identified as a critical variable but we may not have had sufficient variation in information to fully model *Cooling* and be able to infer useful results from permutation testing.

Permutation results for Power and Cooling show less difference between mechanical models. This negative result is still useful in pointing to an underlying weakness in SSCTF modeling in that Power and Cooling were not modeled at a level of detail where wholesale changes in ship configuration actually made much difference. This is unrealistic and could be addressed in future studies.

After concluding the results of permutation, we began presenting the results for algorithm 2. For algorithm 2, the goal was to further enhance permutation by adding the genetic algorithm concept of mutation. We called Algorithm 2 mutation analysis and used it as a way to simulate uncertainty in the manufacturing process. *Mutation analysis also adds to further enhance permutation by helping to identify designs that may not have been considered.*

We begin exploring the results of the enhancement to permutation, mutation analysis, with the critical variable *Space*. It was not surprising that *Space*, as well as the alternative critical variables, possessed a lower percentage chance to pass mutation than permutation. However, despite a lower success rate than permutation, M1 still possessed a 96% chance to survive mutation analysis. It should be noted that the next best performer with regards to mutation analysis on the variable *Space* was M2 with 93% chance to survive testing. In regards to the loss of success rate between permutation and mutation analysis, M2 suffered a 3% loss in performance from its 96% chance to pass permutation testing where M1 only suffered a 1% loss. Between the 97% chance to pass permutation testing and the 96% chance to pass mutation testing, M1 appears to be the better option for a designer most interested in a design's ability to withstand changes to the *Space* critical variable.

Next, we look at the critical variable *Weight* for the mechanical models abilities to withstand mutation analysis. When looking at the critical variable *Weight*, we observed that M1 is most capable of producing a successful "as-built" design, however, M2 was more capable of handling mutation analysis than M1 making M2 the desired mechanical model when looking for the mechanical model most capable (according to these results) of handling potentially unexpected changes to the critical variable *Weight*.

Looking at the result of looking at mutation analysis using the critical variable *Power*, we observed that M1 was better at handling mutation on the critical variable Space than M1, and the M2 was better at handling mutation on the critical variable *Space* than M1. Also, M2 was better than M1 when it came to mutation handling on the critical variable *Weight.* However, while M1 and M2 perform approximately the same when mutation is performed on the critical variable *Power*, M1 performs slightly better than M2 at 99.26% while M2 performs at 98.49%.

M1 is the initial choice model when looking for the mechanical model most capable of handling mutation efficiently on the critical variables *Space* and *Power*, and M2 handles mutation more efficiently on the critical variable *Weight*.

With knowledge of the mechanical models ability to handle mutation on the critical variables, we moved on to mutation on the final critical variable that we performed mutation analysis upon, *Cooling*. The results of mutation on the critical variable *Cooling* were revealing in the sense that mutation had little effect on the success of the critical variable *Cooling.* These results could have been a result of a lack of data for the critical variable *Cooling* or that the result of changes to the critical variable *Cooling* had little impact on the outcome of the testing performed using the 16 metrics.

We concluded the results of mutation on the mechanical model by noting that M1 and M2 both handle mutation well and that the critical variables *Space* and *Weight* are most affected by mutation analysis. Mutation had some effect on the results of testing the 16 metrics on each of the critical variables, but the effects were not very strong. However, the effects of mutation could have stronger impact on an alternative dataset. Lastly, *mutation did help to enhance the effects of permutation on the critical variables by increasing the differences between them.*

In future work, we present a survey of possible additional enhancements beyond the two methods presented herein. Although the SSCTF is generally considered the best frigate level study ever undertaken, additional analysis methods are needed to ensure the best possible ships are developed given known and unavoidable uncertainties in the bid, manufacture and delivery process.

CHAPTER VI – FUTURE WORKS

Cost per Mechanical Model as a Refinement to Permutation and Mutation Analysis

Here we introduce four key ship cost measures (all figures are in Hundreds of Millions of inflation-adjusted Dollars:

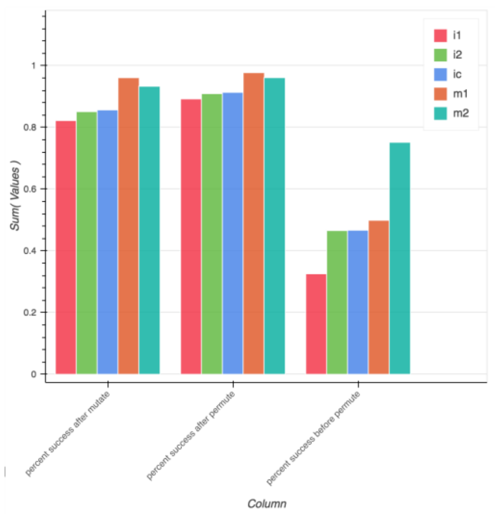
1. *Ship Development Cost:* The initial pre-manufacture cost of research and development including *both research and preparation of a manufacturing process.* This will be higher for ships with novel features (such as non-mechanical drive trains) and is essentially an estimate of how close in manufacturing a new ship line is to an existing ship currently being manufactured,
2. *Ship First Follow Cost:* The cost of the second ship brought out of production - generally the most important key cost metric.
3. *Ship Design Cost - High:* The pessimistic estimate of ship pre-manufacture research only cost. Essentially how novel are a ship's proposed sub-systems.
4. *Ship Design Cost - Low*: The pessimistic estimate of ship pre-manufacture research only cost. Essentially how many ship subsystems can be bought "off the shelf"

Comparison of Ship Mechanical Models as They Relate to Ship Cost

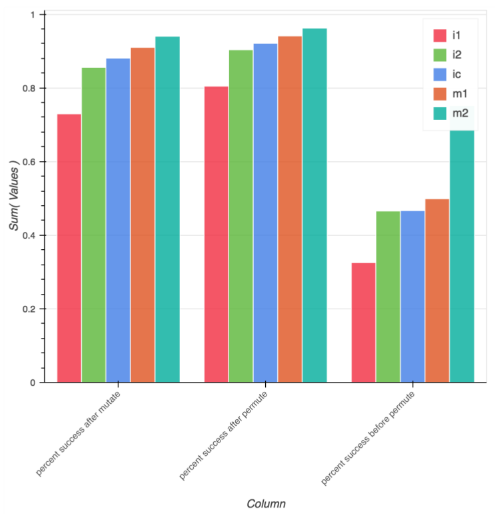
The following is a set of 8 charts showing (1) Space Permutation/Mutation Summary (2) Weight Permutation/Mutation Summary (3) Power Permutation/Mutation Summary and (4) Cooling Permutation/Mutation Summary (5) Ship Development Cost (6) Ship First Follow Cost (7) Ship Design Cost - High and (8) Ship Design Cost - Low.

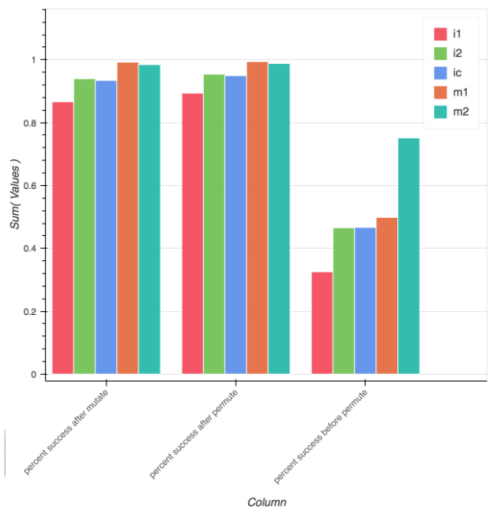
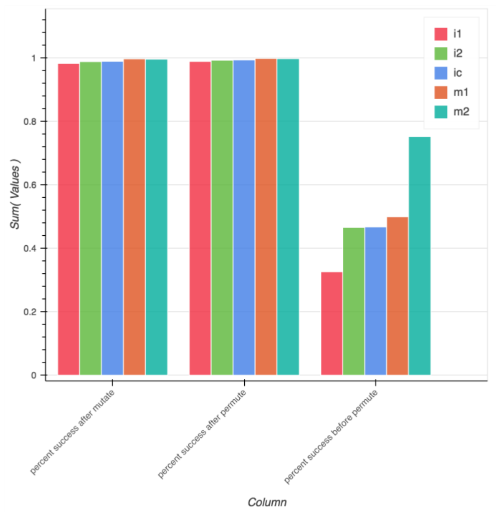
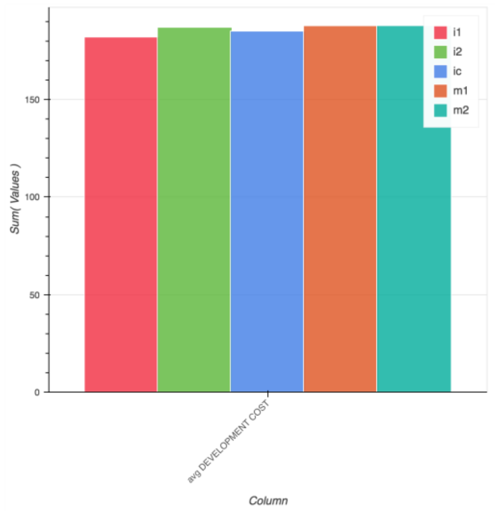
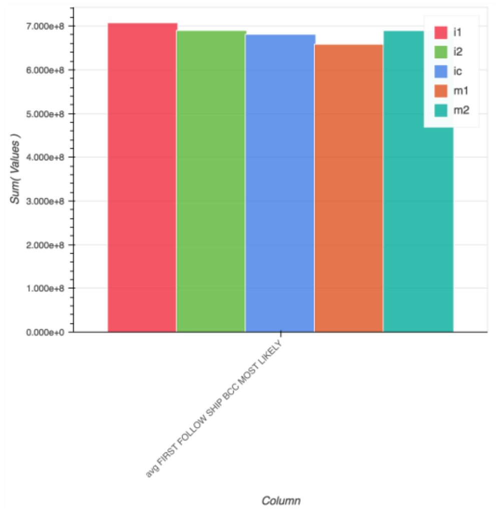
Ship Development Costs are all roughly comparable which we interpret to mean that none of the proposed ships/mechanical models are close to existing ship manufacturing plants. SSCTF was a “clean sheet” design deliberately different from existing Frigate-level ships. Ship First Follow Cost is lowest for M1 representing cost saving of a single-screw design. Ships Design Costs High and Low again show M1 as the least novel design with lowest research cost and most available off the shelf parts.

Space



Weight



Power 	Cooling 
Ship Development Cost 	**Ship First Follow Cost** 
Ship Design Cost - High	**Ship Design Cost - Low**

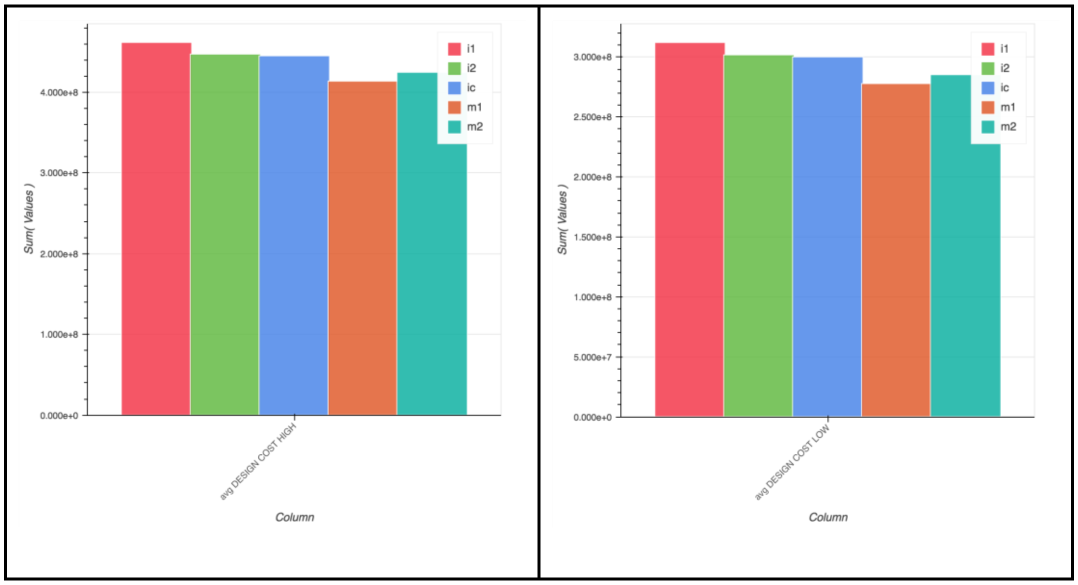


Figure 53. Average statistics permutation, mutation, and average costs

One focus of future work is the possibility that manufacturing risk as characterized by permutation and mutation analysis could be further refined by rerunning cost metrics for ships altered in post-bid design and manufacture to arrive as a "permutation stability per 100 Million Dollars" metric. As shown in the charts above the M1 mechanical Model is significantly cheaper to develop and manufacture. Given that, for the key metric Space, M1 is also more permutation stable as it is possible that a different decision might be made if cost figures were added into risk estimates presented herein. This work was not possible for the current investigation as cost modeling was not available to the ITL SSCTF development team.

Genetic Algorithm Crossover Technique

The genetic algorithm crossover technique begins by taking two parent designs and combining them to produce a child with the best attributes of both parents. It would be interesting to create one child from every pair of parents in the solution space. Look at the feasibility of each of the children and take the standard deviation. Use the standard deviation number to identify all children within the same standard deviation. Children nodes of standard deviation greater than 1 must choose a node greater than 1 to partner with for the next generation. Children that are between standard deviation of .5 to 1 must choose another node in the range of .5 to 1. Last, children within the range of 0 to .5 standard deviation must choose a node in the range of 0 to .5. If a node does not have a partner then that node does not survive to the next generation. Any node produced from a pair of nodes that is an infeasible result is discarded. All nodes start with 1 point of life. After each generation, each node that was able to produce a child that survives feasibility testing gains a point of life. Each node pair that does not produce a feasible child, or cannot pair, loses a point of life. This process could be further divided into more pieces but I would start with .5 increments to the standard deviation. The point of this division of the solution space would cause nodes within ranges of standard deviations to be created. It may result in finding areas of the solution space in which nodes are congregated. It would also reveal nodes that were always able to produce a child with feasible results. It would be interesting to explore the results of this test to see if a notion of node resiliency can be found.

Max and Min Feasibility Impact

Compare the max and min of each key parameter against all other key parameters to determine feasibility impact. Find the region of max and min for each of the key parameters. This will help find regions of acceptable values for each of the key parameters and produce a stronger child by determining their level of impact on each of the other key parameters. It is possible for a variable to have multiple midpoints by having regions of infeasible values contained within the regions of acceptable values. For example, variable A is feasible from 20-30, infeasible from ranges 31-41 and feasible from ranges 42-52. For the variable A, the most likely points of most resilience would be points 25 and 47. This may not be true but it would be interesting to test.

Standard Deviation Distance Plateau Method

Take the objective function fitness score for each of the points in the database and plot those points to a 3d plot where the fitness score is the Z value. Now run a clustering algorithm on the graph and group the points into clusters. For each cluster calculate the standard deviation. Now proceed using 2 different tests based on determining which cluster has the lowest standard deviation score. Test 1 is to look at all points based on cluster and determine which cluster has the lowest standard deviation score. The method will suffer from points that are exceptionally above and below the standard deviation. Test two is to look points within plus or minus 1 standard deviation to determine which cluster has the lowest standard deviation score. By ignoring points that are exceptionally good and bad it will be easier to identify clusters of resilient points. A resilient point, in this case, would not be the individual point but rather a set of points representing the possible feasible choices for the resilient point.

APPENDIX A – Full Source Code: Permutation Stability Analysis

```
author__ = 'James Ross'
#example command line run command with 16 processes
#time mpirun -np 16 python mpiTesting.py
#note there is an excute file that will run this code for C400,P400,S400,criticalVariableValueList
# The file is propFeasPerm
# It runs all permutation testing and takes about 1 day to run.
# Single test runs can be ran in 30 seconds.
from mpi4py import MPI
import sqlite3 as lite
from collections import Counter
import random
import sys
#add the path to the models
sys.path.append("../models/python")
import july11i1 as i1
import july11i2 as i2
import july11ic as ic
import july11m1 as m1
import july11m2 as m2
#add the path to the requirements file
sys.path.append("../analysis/python")
import requirements
#add the path to the metricsv31 file
sys.path.append("../metrics/python")
import metricsv31
from operator import attrgetter
import CONST
import tableTemplate
import os.path
import pandas as pd
import time
import itertools
import numpy as np
import math
import threading
from IPython.core import display as ICD
import plotly.plotly as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use('ggplot')
from bokeh.io import output_notebook, show
import multiprocessing
import bokeh.charts
import bokeh.plotting as bk
bk.output_notebook()
# In[3]:
#if the bat file is used then use the following 4 lines to accept input from the bat file
#permRunsPercent = int(sys.argv[1])
#prop = sys.argv[2]#'SPACE'
#propname = sys.argv[3]#"S400"
#feasible = sys.argv[4]#True for select feasible, False for select infeasible
#combinationChoice = sys.argv[5]
#debug TEsting
# Note: permRuns was changed to permRunsPercent remove this line in final version
permRunsPercent = .10#CONST.PERMRUNS#int(sys.argv[1])
prop = CONST.PROPSPACE
propname = CONST.PROPNAMESPACE
combinationChoice = 0
numDbToTest = 1
percentOfRowsToTest = .10
# set this to true when testing to limit to only one DB to speed things up
#testingApplication = True
testingApplication = True
testingPermuteAgainstMultipleDB = False
```

```
#allow duplicate permutation test
allowDuplicatePermutation = False
# how often to show a row progress update when showing how far along in seed progressing we are
updateRowProgress = 10
criticalVariableValueList = {}
fullCriticalVariableList = {}
geneticDictionary = {}
dfFull ={}
con = Counter()
testOut = 10
testcounter = 0
# run the program using
#  feasible solutions or run it using infeasible solutions
ifFeas = CONST.FEASIBLESOLUTION
# the percentage of the time that we are going to mutate
mutationChance = .5
# a list of the possible primary variables
# this lst is here for reference of what is available and is not actually used anywhere else
FullPrimaryVariableList = {'SPACE',WEIGHT',
                    'POWER','COOLING'}
#a list of the variables we would like to examine
# this can be any subset of FullPrimaryVariableList of 1 to all possible values
PrimaryVariableValueList = {'SPACE'}
#PrimaryVariableValueList = {WEIGHT'}
#PrimaryVariableValueList = {'POWER'}
#PrimaryVariableValueList = {'COOLING'}
csvFilePath = propname+"/"+"permutationWithFeasible_Perm" + str(permRunsPercent) + "_" + propname + "_" + ifFeas +
".csv"
# In[4]:
# create a list of all possible combinations of the primary variable list
# 0 element holds each individual element
# 1 element holds all combinations of values -1 element. This allows comparison of possible combinations
# example access to the combination list
# combinationOptions[ 'option 1 or option 2' ][ 'value in combination' ][ 'combination choice' ]
combinationOptions = []
a = list(itertools.combinations(PrimaryVariableValueList,1))
df = pd.DataFrame(a)[0]
#single column of all elements
combinationOptions.append(df)
combinationOptions
# In[5]:
#multi column of data stored in column row format
a = list(itertools.combinations(PrimaryVariableValueList, len(PrimaryVariableValueList) - 1))
df = pd.DataFrame(a)
df = df.transpose
combinationOptions.append(df)
combinationOptions
# In[6]:
criticalVariableValueList = {}
# In[7]:
testingApplication = True
# In[8]:
# if you would like to see the result of permutation ran on infeasible solutions then use
# the InfeasibleSoultions in the line below and comment out ifFeas in the line above
#ifFeas = "InfeasibleSolutions"
#a simple class for holding table values
class Data:
 #my rank values
 #OVERALL_RANK_BY_FEASIBILITY_PER_HUNDRED_MILLION = 0
 # possibleNumberOfRowsThatCouldHaveBeenTested
 def __init__(self,
BS_CELL,CCA,SEED,NUMTIMES_CCA_FOUND,NUM_TIMES_FAILED_PERMUTE,NUM_VALUES_TESTED_DURING
_PERMUTE,FEASIBLE_AFTER_PERMUTE, AVERAGE_FOLLOW_COST,
          PERCENT_FEASIBLE_AFTER_PERMUTE,UPGRADABILITY_METRIC,UPGRADABILITY_PER_HUNDRED_M
ILLION,
          CHANGE_VULNERABILITY,CHANGE_SPEED,AAW,ASW,SUW,MIW,C2,IO,FEASIBILITY_OF_DESIGN_BY_A
VERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC,
          OVERALLRANK, CCARANK):
```

```
    self.BS_CELL=BS_CELL
    self.CCA=CCA
    self.SEED= SEED
    self.NUMTIMES_CCA_FOUND=NUMTIMES_CCA_FOUND
    self.NUM_TIMES_FAILED_PERMUTE=NUM_TIMES_FAILED_PERMUTE
    self.NUM_VALUES_TESTED_DURING_PERMUTE = NUM_VALUES_TESTED_DURING_PERMUTE
    self.FEASIBLE_AFTER_PERMUTE=FEASIBLE_AFTER_PERMUTE
    self.AVERAGE_FOLLOW_COST=AVERAGE_FOLLOW_COST
    self.PERCENT_FEASIBLE_AFTER_PERMUTE=PERCENT_FEASIBLE_AFTER_PERMUTE
    self.UPGRADABILITY_METRIC=UPGRADABILITY_METRIC
    self.UPGRADABILITY_PER_HUNDRED_MILLION=UPGRADABILITY_PER_HUNDRED_MILLION
    self.CHANGE_VULNERABILITY =CHANGE_VULNERABILITY
    self.CHANGE_SPEED = CHANGE_SPEED
    self.AAW = AAW
    self.ASW = ASW
    self.SUW = SUW
    self.MIW = MIW
    self.C2 = C2
    self.IO = IO
    self.FEASIBILITY_OF_DESIGN_BY_AVERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC=FEASIBILIT
Y_OF_DESIGN_BY_AVERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC
    self.OVERALLRANK = 0
    self.CCARANK = 0
 def __repr__(self):
    return repr(( self.BS_CELL, self.CCA, self.SEED, self.NUMTIMES_CCA_FOUND,
self.NUM_TIMES_FAILED_PERMUTE, self.NUM_VALUES_TESTED_DURING_PERMUTE,
self.FEASIBLE_AFTER_PERMUTE,
              self.AVERAGE_FOLLOW_COST, self.PERCENT_FEASIBLE_AFTER_PERMUTE,
self.UPGRADABILITY_METRIC, self.UPGRADABILITY_PER_HUNDRED_MILLION,
              self.CHANGE_VULNERABILITY,self.CHANGE_SPEED,self.AAW,self.ASW,self.SUW,self.MIW,self.C2,self.I
O,
              self.FEASIBILITY_OF_DESIGN_BY_AVERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC,self
.OVERALLRANK,self.CCARANK))
# In[9]:
NUM_RUNS = CONST.NUMRUNS # (Num Rows in Results DB / Num CCAs) / Num of Seeds
def openReadDbPandas():
 global con, cur, numDBs, dfMain, columnNames
 con = Counter()
 cur = Counter()
 numDBs = 0
 path = '../apd-data/'
 for f in os.listdir(path):
    if os.path.isfile(os.path.join(path, f)):
        if "results" in f:
            numDBs += 1
  # same sql statement for all connections
 sql = 'SELECT * FROM Results WHERE "Req Cumulative" > -1'
  # init the master data frame
 con[0] = lite.connect("../apd-data/results_%d.db" % 0)
 dfMain = pd.read_sql(sql, con[0])
 #columnNames = [description[0] for description in cur[0].description]
 columnNames = list(dfMain)
  # establish a connection to all dbs
 if testingApplication:
    for x in range(1, numDbToTest-1):
        con[x] = lite.connect("../apd-data/results_%d.db" % x)
        dfTemp = pd.read_sql(sql, con[x])
        dfMain = pd.concat([dfMain, dfTemp], axis=0)

    #con[0] = lite.connect("../apd-data/results_%d.db" % 0)
    #dfTemp = pd.read_sql(sql, con[0])
    #dfMain = pd.concat([dfMain, dfTemp], axis=0)
 else:
    for x in range(1, numDBs-1):
        con[x] = lite.connect("../apd-data/results_%d.db" % x)
        dfTemp = pd.read_sql(sql, con[x])
        dfMain = pd.concat([dfMain, dfTemp], axis=0)
 #print (len(dfMain))
 #print (list(dfMain))
# In[10]:
# Testing cell for openReadDbPandas
#this is simply used as a quick populate for testing of functions
openReadDbPandas()
# In[11]:
#def initGlobals():
#    global criticalVariableValueListDf
#    criticalVariableValueListDf = pd.DataFrame()
#    criticalVariableValueList = Counter()
#initGlobals()
#global criticalVariableValueList
#criticalVariableValueList = Counter()
# In[12]:
#maybe ill use this, i dont know
def populateCritVarDictSeedBscellCca(seedName, primaryVar):
 # for every returned row ie every row in the feasible set
 myDf = dfMain.loc[dfMain[CONST.seed] == seedName]
 #for propDict in myDf:
 for index, propDict in myDf.iterrows():
     #Using BS_CELL, CCA, Seed, count number feasible for each [BS_CELL, CCA, and seed] : increment number found
in
     # DataDict[], using all values from propDict, see if they have been added to dataDict yet
     #if this BS_CELL exists in the dictionary then append this row under the BS_CELL
     if propDict[CONST.seed] in criticalVariableValueList:
         if propDict[CONST.BSCELL] in criticalVariableValueList[propDict[CONST.seed]]:
             if propDict[CONST.CCA] in criticalVariableValueList[propDict[CONST.seed]][propDict[CONST.BSCELL]]:
                 criticalVariableValueList[propDict[CONST.seed]][propDict[CONST.BSCELL]][propDict[CONST.CCA]].append(p
ropDict[primaryVar])
             else:
                 criticalVariableValueList[propDict[CONST.seed]][propDict[CONST.BSCELL]][propDict[CONST.CCA]] =
[propDict[primaryVar]]
         else:
             criticalVariableValueList[propDict[CONST.seed]][propDict[CONST.BSCELL]] =
{propDict[CONST.CCA]:[propDict[primaryVar]]}
     else:
         criticalVariableValueList[propDict[CONST.seed]]= {propDict[CONST.BSCELL]:  {propDict[CONST.CCA]:
[propDict[primaryVar]]}}
# In[13]:
# count the number of values in a dataframe
# convert the return to a dataframe example: pd.DataFrame(lengthCounter)
def populateLengthDictionary(seedName):
 print("populating length dictionary from full data set for seed %s" % (seedName))
 lengthCounter = {}
 myDf = dfFull.loc[dfFull[CONST.seed] == seedName]
 for index, propDict in myDf.iterrows():
     #Using BS_CELL, CCA, Seed, count number feasible for each [BS_CELL, CCA, and seed] : increment number found
in
     # DataDict[], using all values from propDict, see if they have been added to dataDict yet
     #if this BS_CELL exists in the dictionary then append this row under the BS_CELL
     if propDict[CONST.seed] in lengthCounter:
         if propDict[CONST.BSCELL] in lengthCounter[propDict[CONST.seed]]:
             lengthCounter[propDict[CONST.seed]][propDict[CONST.BSCELL]] += 1
         else:
             lengthCounter[propDict[CONST.seed]][propDict[CONST.BSCELL]] = 1
     else:
         lengthCounter[propDict[CONST.seed]] = {propDict[CONST.BSCELL]: 1}
 # lengthDf = pd.DataFrame(lengthCounter)
 print ("Finished populating length dictionary from full data set for seed %s.  It will take some time to finish the pickling of
data fo this process." % (seedName))
 return lengthCounter
# In[14]:
#populate a dataframe using the list of values for each bscell.
#Use the unique list of values created by populateUniqueValueDf
# The purpose is to know if a value has been tested for a bscell in a specific seed.
def populateTestedValuesDictionary(seedName, primaryVar):
 global testedValuesDf
 testedCounter = criticalVariableValueList
 #-->
 for seed in criticalVariableValueList:
     for bscell in criticalVariableValueList[seed]:
         for cca in criticalVariableValueList[seed][bscell]:
             #for curCCAVal in range(0, len(criticalVariableValueList[seed][bscell][cca])):
             #if isinstance(testedCounter[seed][bscell][cca][curCCAVal], dict) == False:
             if isinstance(testedCounter[seed][bscell][cca][0], dict) == False:
                 testedCounter[seed][bscell][cca] = pd.Series(testedCounter[seed][bscell][cca]).unique()
             # for val in testedCounter[seed][bscell][cca] :
             tmpAry = testedCounter[seed][bscell][cca]
             aryObj = []
             for val in tmpAry:
                  # i noticed that dictionaries were rea
                 if isinstance(val, dict) == False:
                     aryObj.append({'value': val, 'tested': False})
                 else: aryObj.append({'value': val['value'], 'tested': False})
                 # store the ary of value and weather the value has been tested back to the bscell
             testedCounter[seed][bscell][cca] = aryObj
 #testedCounter['m2']['1F']['1F.CS24']
 testedValuesDf = pd.DataFrame(testedCounter)
# A BSCELL may not necessarily occur in all seeds as a feasible design
# In[15]:
testcounter=0
def getRowData(curRow):
 # create a key value pair of this row's values
 colNum = 0
 rowDict = {}
 columnHeadings = list(dfMain)
 columnHeadings.append("RowID")
 for col in columnHeadings:
     rowDict[col] = curRow[colNum]
     colNum += 1
 myDataDict = {}
     # determine how many values we have
 numTimesToPermute = 0
 localVarValList = criticalVariableValueList
 # figure out how many values we have in this seed the first time we encounter it
 if (rowDict[CONST.seed] in permutePerSeed):
     numTimesToPermute = permutePerSeed[rowDict[CONST.seed]]
 else:
     for bscell in localVarValList[rowDict[CONST.seed]]:
         numTimesToPermute += len(localVarValList[rowDict[CONST.seed]][bscell])
     # keep track of how many values were tested for this seed
     permutePerSeed[rowDict[CONST.seed]] = numTimesToPermute
 # apply the percentage modifier in case the user wants to use less rows than all possible
 numTimesToPermutePercent = int(math.ceil(numTimesToPermute * permRunsPercent))
 # keep track of how many values were tested for this seed
 #totalNumberOfValuesTestedForSeed += numTimesToPermutePercent
 # if there was only 1 value then test
 if (numTimesToPermutePercent < 1 and numTimesToPermutePercent > 0):
     numTimesToPermutePercent = 1
 #if (curRowProgress % updateRowProgress == 0):
 #    print("Currently on row %d of %d possible rows for this seed %s. I am process number %d" % (
 #    curRowProgress, totalRows, rowDict[CONST.seed], RANK))
 #    print("About to test %d possible values for this row" % (int(math.ceil(numTimesToPermutePercent))))


     # if you didnt check all the values then go ahead, else all values for this bscell have been tested
     #  numTimesToPermute: number of possible values for this row
 for permute in range(0, numTimesToPermutePercent):
     firstIndexOfUntestedSeed = ""
     firstIndexOfUntestedBscell = ""
     firstIndexOfUntestedCca = ""
     firstIndexOfUntestedCcaVal = -1
     numUntested = 0
     numTested = 0
     firstIndexOfUntested = -1
     # For the very first row, set up the myDataDictionary
```

```
    if permute == 0:
        saveprop = rowDict[primaryVar]
        # Using BS_CELL,CCA, seed, count number feasible for each [BS_CELL,CCA,seed] : increment number
        #  found in myDataDict[], using all values from rowDict, See if they have been added to myDataDict yet
        if (rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]) in myDataDict:
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
                0] += 1  # count 0 -  #numTimesCcaFound
            # everytime this CCA is reencountered, add to the total of values tested
            # myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][15] +=
numTimesToPermute
        else:
            # Initialize this [BS_CELL, CCA,seed] in the myDataDict by adding required info
            # [0=number feasible, 1=number tested, 2=number still feasible after permutation,
            # 3=average cost running total] notify that this node is done working
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]] = [1, 0, 0, 0, 0, 0, 0, 0, 0,
                                                                                      0, 0, 0, 0, 0, 0]
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][3] = str(
                rowDict[CONST.METRIC_AVERAGE_FOLLOW_END_COST_MOST_LIKELY])
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][6] = str(
                rowDict[CONST.METRIC_VULNERABILITY])
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][7] = str(
                rowDict[CONST.DESIGN_SUSTAINED_SPEED])
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][8] = (
                rowDict[CONST.AAW]).strip(' ')
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][9] = (
                rowDict[CONST.ASW]).strip(' ')
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][10] = (
                rowDict[CONST.SUW]).strip(' ')
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][11] = (
                rowDict[CONST.MIW]).strip(' ')
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][12] = (
                rowDict[CONST.C2]).strip(' ')
            myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][13] = (
                rowDict[CONST.IO]).strip(' ')
            # If there was a value that needed to be tested then run regression and things on this value
            # if firstIndexOfUntestedCcaVal != -1:
            # Attempt to randomly get the index of a value in the critical variable list
    chosenSeed = rowDict[CONST.seed]
    chosenbscell = ""
    chosenCCA = ""
    numBscell = len(localVarValList[chosenSeed])
    chosenBscellIndex = random.randint(0, numBscell - 1)
    curBscellIndex = 0
    chosenCcaValIndex = 0
    for bscell in localVarValList[chosenSeed]:
        if curBscellIndex == chosenBscellIndex:
            chosenbscell = bscell
            numCca = len(localVarValList[chosenSeed][chosenbscell])
            chosenCcaIndex = random.randint(0, numCca - 1)
            curCcaIndex = 0
            for cca in localVarValList[chosenSeed][chosenbscell]:
                if curCcaIndex == chosenCcaIndex:
                    # chosenCCA = localVarValList[chosenSeed][chosenbscell][cca]
                    chosenCCA = cca
                    numValForCca = len(localVarValList[chosenSeed][chosenbscell][chosenCCA])
                    chosenCcaValIndex = random.randint(0, numValForCca - 1)
                    break
                else:
                    curCcaIndex += 1
            break
        else:
            curBscellIndex += 1
        # Once we have attempted to randomly choose a value to try for this row, make sure we have a
        # random index of a value that hasn't been tested so that we can meet our percentage.
        # If we don't get a random number that hasn't been tested, take the next number that hasn't been tested
    if (localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['tested'] == True):
        seed = rowDict[CONST.seed]
        for bscell in localVarValList[rowDict[CONST.seed]]:
            for cca in localVarValList[seed][bscell]:
                for curCCAVal in range(0, len(localVarValList[seed][bscell][cca])):
                    if (localVarValList[seed][bscell][cca][curCCAVal]['tested'] == False):
                        if firstIndexOfUntestedCcaVal == -1:
                            # store a reference into the structure to the first untested value
                            firstIndexOfUntestedSeed = seed
                            firstIndexOfUntestedBscell = bscell
                            firstIndexOfUntestedCca = cca
                            firstIndexOfUntestedCcaVal = curCCAVal
                        numUntested += 1
                    else:
                        numTested += 1
        # How often do we state our progress
        '''
        if (permute % (int(math.ceil(numTimesToPermutePercent)) * .10) == 0):
            if (numUntested == 0):
                # print("No values left to test for this row. Adding to counter number of times a valid value has appeared")
                a = 0
            else:
                print( "Untested values for bscell %s in Seed %s: %d. Tested Values for this bscell: %d. Process %d" % (
                        rowDict[CONST.BSCELL], rowDict[CONST.seed], numUntested, numTested, RANK))
                #print(
                #"Untested values for bscell %s in Seed %s: %d. Tested Values for this bscell: %d. Process %d row %d" % (
                #    rowDict[CONST.BSCELL], rowDict[CONST.seed], numUntested, numTested, RANK, curRowProgress))
                # End of print message
        '''
        randomPermute = \
        localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][firstIndexOfUntestedCca][
            firstIndexOfUntestedCcaVal]['value']
        # if we are allowing duplicate tests of the same value
        localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][firstIndexOfUntestedCca][
            firstIndexOfUntestedCcaVal]['tested'] = True
    else:
        randomPermute = localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['value']
        # if we are allowing duplicate tests of the same value
        if allowDuplicatePermutation == False:
            localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['tested'] = True
            # Assign the randomly or next chosen value to the critical var in prop dict
    rowDict[primaryVar] = randomPermute
    # run the datarow through the appropriate regression model for this seed
    if rowDict[CONST.seed] == CONST.i1:
        rowDict = i1.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.i2:
        rowDict = i2.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.ic:
        rowDict = ic.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.m1:
        rowDict = m1.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.m2:
        rowDict = m2.RegEx(rowDict)
        # run requirements and metrics on the datarow
    rowDict = requirements.RegEx(rowDict)
    rowDict = metricsv31.RegEx(rowDict)
    '''
    0 numTimes_Cca_Found = float(count[0])
    1  self.NUM_TIMES_FAILED_PERMUTE = NUM_TIMES_FAILED_PERMUTE
    2  feasible_after_permute = float(count[2])
    14 NUM_POSSIBLE_VALUES = float(count[15])
    '''
    # number of possible values
    myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][14] += 1
    # NOTE: For both the case where a row has been tested and not been tested, increment the appropriate values in the
myDataDict
    # If row is feasible after permute then add to still feasible count and to numtested
    if rowDict[CONST.REQ_CUMULATIVE] > -1:
        # This is a really interesting value. If the row fails feasibility test after permutation then it means that it lacks
resilience
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
            2] += 1  # count 2 - # feasible after permute
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][4] += rowDict[
            CONST.METRIC_VULNERABILITY]
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][5] += rowDict[
            CONST.DESIGN_SUSTAINED_SPEED]
    else:
        # This means the row is no longer feasible so only add to number of rows tested
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
            1] += 1  # count 1 - #NUM_TIMES_FAILED_PERMUTE
        # Another row has completed
 #curRowProgress += 1
 totalNumUntested = 0
 totalNumtested = 0
 seed = rowDict[CONST.seed]
 # Reset the values for each row so that each row has fair access to possible values
 for bscell in localVarValList[rowDict[CONST.seed]]:
    for cca in localVarValList[seed][bscell]:
        for curCCAVal in range(0, len(localVarValList[seed][bscell][cca])):
            # While values are being reset, keep track of how many rows have not been tested and
            # also track the number of values that were tested for this seed
            if (localVarValList[seed][bscell][cca][curCCAVal]['tested'] == False):
                totalNumUntested += 1
            else:
                totalNumtested += 1
            localVarValList[seed][bscell][cca][curCCAVal]['tested'] = False


 if(int(rowDict['RowID']) % 50 == 0):
    print ("Computed seed %s bascel %s cca %s. Using row %s" % (rowDict[CONST.seed],rowDict[CONST.BSCELL],
rowDict[CONST.CCA],rowDict['RowID']))
 #for item in myDataDict:
 #    dataDict[item] = myDataDict[item]
 return {'seed':rowDict[CONST.seed],'bscell': rowDict[CONST.BSCELL],'cca': rowDict[CONST.CCA],'myDataDict':
myDataDict, 'totalNumUntested': totalNumUntested, 'totalNumTested': totalNumtested}
# In[16]:
#%success for mechanical model
def initRowsAndPermuteForSeedSuccessPercentage(seedName, primaryVar):
 global row, dictrows, propDict, permute, saveprop, randomPermute,workQueue,permutePerSeed
  # total number of rows that were tested by this process and the number of values left untested of the known
  #  possible feasible values
 totalNumUntested = 0
 totalNumtested = 0
 totalNumberOfValuesTestedForSeed = 0
 myDf = dfMain.loc[dfMain[CONST.seed] == seedName]
 curRowProgress = 0
 totalRows = len(myDf)
 permutePerSeed = {}
  # number of rows to work on per process
 if SIZE > totalRows:
    numProcessRequired = totalRows
 else: numProcessRequired = SIZE
 # Minimum of 1 process per row
 if RANK <= numProcessRequired:
    # This is the total number of rows this process will work on
    myRowCount = int(math.ceil(totalRows / numProcessRequired))
    if( myRowCount > 1):
        #if RANK == 0:
        #    startIndex = 0
        #else:
        startIndex = (RANK * myRowCount) #- 1
        endIndex = ((RANK + 1) * myRowCount) - 1
         # for the last set of rows, make sure we don't overshoot the number of rows
        #if( endIndex > totalRows):
        #    endIndex = totalRows - 1
    elif myRowCount == 1:
        startIndex = RANK
        endIndex = RANK
    else:
        startIndex = 0
        endIndex = 0
    #The row in the database this process is starting on
    print ( "Rank %d startIndex %d endIndex %d Number of processes %d Row count %d" % (RANK, startIndex,
endIndex, SIZE, myRowCount))
    columnHeadings = list(dfMain)
     # start on the proper row for this process
    curRowProgress = startIndex
    endIndex = startIndex + ((endIndex-startIndex) * percentOfRowsToTest)
    if( endIndex < 1):
        endIndex = 1
    # Create a thread for every row. the thread will handle permute for that row and return stats for the row.
    threadList = []
    rowList = []
    #rowList = {}
    threadCount = startIndex
    for curRow in myDf.iloc[int(startIndex):int(endIndex)].values:
        #threadList.append("Thread-" + str(threadCount))
        #rowList.append(curRow)
        curRow = np.append(curRow,str(threadCount))
        rowList.append(curRow)
        threadCount += 1
    print ("Computing permute with maximum power")

    print("Expect the rows to print in the order they are processed.")
    #test = getRowData(rowList[0])
    numThreads = int(endIndex) - int(startIndex)
    test1 = multiprocessing.cpu_count()
    dataList = []
    ##############################################
    # non parallel way
    #for row in rowList:
    #    dataList.append(getRowData(row))
    ##############################################
    ####Paralle way
    #
    pool = multiprocessing.Pool(multiprocessing.cpu_count())
    dataList = pool.map(getRowData, rowList)
    pool.close()
    pool.terminate()
    pool.join()
    ##############################################
    print("Finished permute")
    for item in dataList:
        seed = item['seed']
        bscell = item['bscell']
        cca = item['cca']
        if (seed, bscell,cca) in dataDict:
            dataDict[seed,bscell,cca][0] += item['myDataDict'][seed,bscell,cca][0]
            dataDict[seed, bscell, cca][1] += item['myDataDict'][seed, bscell, cca][1]
            dataDict[seed, bscell, cca][2] += item['myDataDict'][seed, bscell, cca][2]
            dataDict[seed, bscell, cca][14] += item['myDataDict'][seed, bscell, cca][14]
        else : dataDict[seed,bscell,cca]= item['myDataDict'][seed,bscell,cca]
    print ("Exiting Main Thread")
# In[17]:
test =""
# Start open and grab all rows with ReqCumulative > -1
def mpiCreateDictionaryAndPermute():
 global mpi_comm, SIZE, RANK, ROOT, dataDict, criticalVariableValueList,primaryVar,  e
 mpi_comm = MPI.COMM_WORLD
 SIZE = mpi_comm.Get_size()
 RANK = mpi_comm.Get_rank()
 ROOT = 0
 dataDict = Counter()



 try:
    print ("Node " + str(RANK) + " Reading DBs")
```

```
        dbReadTime = time.time()
        openReadDbPandas()
        print ("Process %d reading databases: %s seconds " % (RANK, time.time() - dbReadTime))
         # Get the list of seeds
        seedList = dfMain.seed.unique()
         # For each seed family of designs
        for seedName in seedList:
            if combinationChoice == 0:
                for primaryVar in combinationOptions[0]:
                    print ("Beginning work on seed family %s using primary variable %s" % ( seedName, primaryVar))
                     # Print column heading
                    print ("Node " + str(RANK) + " beginning work on seed family " + seedName)
                     #populate the data each processor will work on
                    print ("Node " + str(RANK) + " configuring data structures")
                    configTime = time.time()
                    ########################################
                    ############# Testing #################
                    populateCritVarDictSeedBscellCca(seedName, primaryVar)
                    print ("Process %d configuring data structures: %s seconds " % (RANK, time.time() - configTime))
                     # go ahead and make a easy to reference dataframe that tells me the length of each element
                    #populateLengthDictionary(seedName, primaryVar)
                     # make a list of which values have been tested for each BSCELL
                    populateTestedValuesDictionary(seedName, primaryVar)
                     # run the substitution algorithm testing for resilience
                    print ("Node " + str(RANK) + " beginning permutation")
                    permuteTime = time.time()
                    initRowsAndPermuteForSeedSuccessPercentage(seedName, primaryVar)
                    print ("Process %d time it took for permute: %s seconds " % (RANK, time.time() - permuteTime))
            elif combinationChoice == 1:
                for primaryVar in combinationOptions[1]:
                    print ("Beginning work on seed family %s using primary variable %s " % ( seedName, primaryVar))
                     #populate the data each processor will work on
                    print ("Node " + str(RANK) + " configuring data structures")
                    configTime = time.time()
                    populateDictionary(seedName)
                    print ("Process %d configuring data structures: %s seconds " % (RANK, time.time() - configTime))
                     #run the substitution algorithm testing for resilience
                    print ("Node " + str(RANK) + " beginning permutation")
                    permuteTime = time.time()
                    initRowsAndPermuteBroken(seedName)
                    print ("Process %d time it took for permute: %s seconds " % (RANK, time.time() - permuteTime))
    except lite.Error as e:
        # report errors if they occur
        print ("Error retrieving data for permutation test")
        print ("Error: %s" % e)
        exit(1)
#Gather the results from each node into one dictionary
def getFeasibilityOfDesignByAverageFollowCostPerHundredMillionMetric(statData):
 return statData.FEASIBILITY_OF_DESIGN_BY_AVERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC
def getCCA(statData):
 return statData.CCA
# In[18]:
######################## Untested functions #############################################
def populateFinalDict():
 global dict, item
 for dict in dataDict:
    if dict in finalDict:
        finalDict[dict][0] += dataDict[dict][0]
        finalDict[dict][1] += dataDict[dict][1]
        finalDict[dict][2] += dataDict[dict][2]
        finalDict[dict][3] += dataDict[dict][3]
        finalDict[dict][4] += dataDict[dict][4]
        finalDict[dict][5] += dataDict[dict][5]
        finalDict[dict][6] += dataDict[dict][6]
        finalDict[dict][7] += dataDict[dict][7]
        finalDict[dict][8] += dataDict[dict][8]
        finalDict[dict][9] += dataDict[dict][9]
        finalDict[dict][10] += dataDict[dict][10]
        finalDict[dict][11] += dataDict[dict][11]
        finalDict[dict][12] += dataDict[dict][12]
        finalDict[dict][13] += dataDict[dict][13]
        finalDict[dict][14] = dataDict[dict][14]
    else:
        finalDict[dict] = [dataDict[dict][0], dataDict[dict][1], dataDict[dict][2], dataDict[dict][3], dataDict[dict][4],
                        dataDict[dict][5],
                        dataDict[dict][6], dataDict[dict][7], dataDict[dict][8], dataDict[dict][9], dataDict[dict][10],
                        dataDict[dict][11], dataDict[dict][12], dataDict[dict][13], dataDict[dict][14]]
# In[19]:
#Add all metrics and feasibility calculations to the data
def addMetricsAndFeasibilityToDataRows():
 global keylist, count, feasible_after_permute, numTimes_Cca_Found, num_times_failed_permute, avgcost,
changevulnerability, changespeed,         percentFeasible, percentFeasibleAfterPermute, upgradeMetric,
metricPerHundredMillion,         feasibilityOfDesignByAvgFollowCostPerHundredMillionMetric
 myTableTemplate = tableTemplate
 # countf1 = 0
 for keylist, count in finalDict.items():
    '''
    1  self.NUM_TIMES_FAILED_PERMUTE = NUM_TIMES_FAILED_PERMUTE
    2  feasible_after_permute = float(count[2])
    14 NUM_POSSIBLE_VALUES = float(count[15])
    '''
    # Num rows feasible after permuatation testing
    feasible_after_permute = float(count[2])
    # original number of rows passing ReqCumulative test
    numTimes_Cca_Found = float(count[0])
    # Original num rows passing req cumulative test by permutation swapping
    #num_tested = float(num_feasible) * float(permRuns)
    #num_tested = float(num_feasible) * float(count[14])
    num_times_failed_permute = float(count[1])
    NUM_POSSIBLE_VALUES = float(count[14])
    # avg follow cost of ship rows
    avgcost = float(count[3])
    if feasible_after_permute > 0:
        changevulnerability = -float(count[6]) + (float(count[4])/float(feasible_after_permute))
        changespeed = -float(count[7]) + (float(count[5])/float(feasible_after_permute))
    else:
        changevulnerability = 0
        changespeed = 0
    #TODO: THIS REQUIRES having data from all databases and figuring out how many feasible values this cca
has
    percentFeasible = 1
    #percentFeasible = float(numTimes_Cca_Found)/ float((NUM_RUNS))
    percentFeasibleAfterPermute = float(feasible_after_permute)/float(NUM_POSSIBLE_VALUES)
    if ( num_times_failed_permute == 0):
        upgradeMetric = percentFeasible * feasible_after_permute / 1
    else: upgradeMetric = percentFeasible * feasible_after_permute/num_times_failed_permute
    metricPerHundredMillion = upgradeMetric * hundredMillion/ avgcost
    feasibilityOfDesignByAvgFollowCostPerHundredMillionMetric = ((percentFeasible *
percentFeasibleAfterPermute)/avgcost) * hundredMillion
    statData.append(
        Data(
            str(keylist[0]),# BS CELL - combat capability
            str(keylist[1]),# CCA combat capability alternative
            str(keylist[2]),# SEED
            str(numTimes_Cca_Found),# numTimes_Cca_Found
            str(num_times_failed_permute),#num_times_failed_permute
            str(NUM_POSSIBLE_VALUES),#NUM_POSSIBLE_VALUES
            str(feasible_after_permute),#feasible after permute
            str(avgcost),#average follow cost
            str(percentFeasibleAfterPermute),#percent still feasible
            str(upgradeMetric),#upgrade metric
            str(metricPerHundredMillion),#metric per hundred million
            str(changevulnerability),#change vulnerability
            str(changespeed),#change speed
            count[8],#AAW
            count[9],#ASW
            count[10],#SUW
            count[11],#MIW
            count[12],#C2
            count[13],#IO
            str(feasibilityOfDesignByAvgFollowCostPerHundredMillionMetric),#Feasibility of Design By Average Follow Cost
Per HundredMillion Metric
            0,#Overall Rank
            0#CCA Rank
        )
    )
# In[20]:
def addOverallRankAndSortData():
 global ovrRank, statAry, getRank, total1f, x
 # Add overall rank
 ovrRank = 0
 statAry = []
 # get data sorted by key then sorted by the BSCELL then sorted by overallRank
 getRank = sorted(statData, key=getFeasibilityOfDesignByAverageFollowCostPerHundredMillionMetric, reverse=True)
 total1f = 0.0
 for x in getRank:
     # label each row by rank
     x.OVERALLRANK = ovrRank
     ovrRank = ovrRank + 1
# In[21]:
def addCCARankAndSortData():
 global ccaRankCount, curCell, ccaRank, firstPass, getCCARank, x, sortedBSCELL, y
 ###Add rank within CCA
 ccaRankCount = 0
 curCell = ""
 ccaRank = []
 # initialize a new BSCELL on first pass only
 firstPass = True
 # after overall rank has been assigned the sort by BSCELL in order to get Ready to assign Rank to Each CCA
 getCCARank = sorted(getRank, key=attrgetter(CONST.BS_CELL), reverse=False)
 for x in getCCARank:
     if firstPass == True:
         curCell = x.BS_CELL
         firstPass = False
         # We have hit a new set of BSCELL and need to sort the previous BSCELL list by the overallRank
         # and then assign a CCA rank based on who has the best overall rank. The list are being modified
         # by the reference so change made to sorted BSCELL after statData
     if curCell != x.BS_CELL:
         ccaRankCount = 0
         #for x in ccaRank:
         sortedBSCELL = sorted(ccaRank, key=attrgetter(CONST.OVERALLRANK), reverse=False)
         for y in sortedBSCELL:
             y.CCARANK = ccaRankCount
             ccaRankCount = ccaRankCount + 1
         ccaRank = []
     ccaRank.append(x)
     curCell = x.BS_CELL
# In[22]:
#sort the data by 'x' then 'j' then 'a'. any three values could go here
def addSeedRankAndSortData():
 global seedRankCount, curCell, seedRank, firstPass, getSeedRank, finalList, x, sortedBSCELL, y
 ###Add rank for each seed
 seedRankCount = 0
 curCell = ""
 seedRank = []
 firstPass = True
 # get list sorted on feas per mil metric and then sort based on seed
 getSeedRank = sorted(getRank, key=attrgetter(CONST.SEED), reverse=False)
 finalList = []
 # add seed 1 - (n-1) to the list and when were done add the final seed
 for x in getSeedRank:
     if firstPass == True:
         curCell = x.SEED
         firstPass = False
     if curCell != x.SEED:
         seedRankCount = 0
         sortedBSCELL = sorted(seedRank, key=attrgetter(CONST.OVERALLRANK), reverse=False)
         for y in sortedBSCELL:
             y.seedRank = seedRankCount
             finalList.append(y)
             seedRankCount = seedRankCount + 1
         seedRank = []
     seedRank.append(x)
     curCell = x.SEED
 #add the final seed
 seedRankCount = 0
 sortedBSCELL = sorted(seedRank, key=attrgetter(CONST.OVERALLRANK), reverse=False)
 for y in sortedBSCELL:
     y.seedRank = seedRankCount
     finalList.append(y)
     seedRankCount = seedRankCount + 1
 seedRank = []
# In[23]:
def writeDataToCSVFile():
 global y, varAry, stat_text, handle
  #print finalDict
 stat_text="SEED, BS CELL,CCA, NUM TIMES CCA FOUND, NUM TIMES FAILED PERMUTE,NUM VALUES TESTED
DURING PERMUTE,FEASIBLE AFTER PERMUTE,PERCENT FEASIBLE AFTER PERMUTE,"
 stat_text+="AVERAGE FOLLOW COST (Original Design),'Upgradabiliy' METRIC,'Upgradability' PER $100
Million,Change Vulnerability,"
 stat_text+="Speed,AAW,ASW,SUW,MIW,C2,IO,Feasibility of Design By Average Follow Cost Per HundredMillion
Metric,"
 stat_text+="Overall Rank,CCA Rank,Seed Rank\n"
 for y in finalList:
     varAry = str(
         y.BS_CELL + "," + y.CCA + "," + y.SEED + "," + y.NUMTIMES_CCA_FOUND + "," +
y.NUM_TIMES_FAILED_PERMUTE + "," +
         y.NUM_VALUES_TESTED_DURING_PERMUTE + "," + y.FEASIBLE_AFTER_PERMUTE + "," +
y.PERCENT_FEASIBLE_AFTER_PERMUTE + "," +
         y.AVERAGE_FOLLOW_COST + "," + y.UPGRADABILITY_METRIC + "," +
y.UPGRADABILITY_PER_HUNDRED_MILLION + "," + y.CHANGE_VULNERABILITY + "," +
         y.CHANGE_SPEED + "," + y.AAW + "," + y.ASW + "," + y.SUW + "," + y.MIW + "," + y.C2 + "," + y.IO + "," +
         y.FEASIBILITY_OF_DESIGN_BY_AVERAGE_FOLLOW_COST_PER_HUNDRED_MILLION_METRIC + "," +
str(y.OVERALLRANK) +
         "," + str(y.CCARANK) + " , " + str(y.seedRank))
     stat_text += varAry + "\n"
 handle = open(
     propname + "/" + "permutationWithFeasible_Perm" + str(permRunsPercent) + "_" + propname + "_" + ifFeas + ".csv",
"w")
 handle.write(stat_text)
 handle.close()
# Once you have run everything above has been run at least once, you may begin exploring the data in the cells below
# In[24]:
#csvFilePath
# In[25]:
##################################################################
##############Begin gathering results#########################
###Step 1: Determine seed success before permutation############
#Read in the whole data set
def readEntireDataset():
 global dfFull
 dfFull = pd.DataFrame()
 con = Counter()
 cur = Counter()
 numDBs = 0
 path = '../apd-data/'
 for f in os.listdir(path):
     if os.path.isfile(os.path.join(path, f)):
         if "results" in f:
             numDBs += 1
 ############
 dfTemp = []
 # Attempted Parallel read of all databases. Does not work##
 #  DO NOT DELETE, SAVE FOR REFERENCE  #
 '''
 localPool = multiprocessing.Pool(multiprocessing.cpu_count())
  # if were testing just do 3 databases
 if testingApplication == True:
      # Read in all data sets
     dfTemp = localPool.map(getAllDataFromDbWithConnecting, range(0, numDbToTest))
 else:
      # Read in all data sets
     dfTemp = localPool.map(getAllDataFromDbWithConnecting, range(0, numDBs-1))
 localPool.close()
 #localPool.terminate()
 localPool.join()
 '''
 #Non parallel read of all databases##
 #  DO NOT DELETE, SAVE FOR REFERENCE  #
  # initialize
 dfTemp.append( getAllDataFromDbWithConnecting(0))
 if testingApplication:
     for x in range(1, numDbToTest):
         dfTemp.append(getAllDataFromDbWithConnecting(x))
         print( "finished reading db %d" % (x))
 else:
     for x in range(1, numDBs-1):
         dfTemp.append(getAllDataFromDbWithConnecting(x))
         print( "finished reading db %d" % (x))
 ########################################
 dfFull = dfTemp[0]
 for index in range(1,len(dfTemp)):
     dfFull = pd.concat([dfFull, dfTemp[index]], axis=0)
 print("Finished reading in entire data set")
# In[26]:
# Since we are reading the entire data set, this needs to happen in parallel
def getAllDataFromDbWithConnecting(dbNum):
 myCon = lite.connect("../apd-data/results_%d.db" % dbNum)
  # same sql statement for all connections
 sql = 'SELECT * FROM Results'
 myDataFrame = pd.read_sql(sql, myCon)
 print("Finished reading data for db %d. Expect the pickling of this data to take several minutes based on the size of the
db" % (dbNum))
 return myDataFrame
# In[27]:
# Since we are reading the entire data set, this needs to happen in parallel
def getAllDataFromDbWithoutConnecting(dbNum):
 myCon = lite.connect("../apd-data/results_%d.db" % dbNum)
  # same sql statement for all connections
 sql = 'SELECT * FROM Results'
 myDataFrame = pd.read_sql(sql, myCon)
 return myDataFrame
# In[28]:
# populate the fullCriticalVariableList
# this is the same thing as criticalVariableList except it is for the whole data set
def populateFullValueList(seedName):
# for every returned row ie every row in the feasible set
myDf = dfFull.loc[dfFull[CONST.seed] == seedName]
for index, propDict in myDf.iterrows():
    #Using BS_CELL, CCA, Seed, count number feasible for each [BS_CELL, CCA, and seed] : increment number found
in
    # DataDict[], using all values from propDict, see if they have been added to dataDict yet
    #if this BS_CELL exists in the dictionary then append this row under the BS_CELL
    if propDict[CONST.seed] in fullCriticalVariableList:
        if propDict[CONST.BSCELL] in fullCriticalVariableList[propDict[CONST.seed]]:
            if propDict[CONST.CCA] in fullCriticalVariableList[propDict[CONST.seed]][propDict[CONST.BSCELL]]:
                fullCriticalVariableList[propDict[CONST.seed]][propDict[CONST.BSCELL]][propDict[CONST.CCA]].append(prop
Dict[primaryVar])
            else:
                fullCriticalVariableList[propDict[CONST.seed]][propDict[CONST.BSCELL]][propDict[CONST.CCA]] =
[propDict[primaryVar]]
        else:
            fullCriticalVariableList[propDict[CONST.seed]][propDict[CONST.BSCELL]] =
{propDict[CONST.CCA]:[propDict[primaryVar]]}
    else:
        fullCriticalVariableList[propDict[CONST.seed]]= {propDict[CONST.BSCELL]:  {propDict[CONST.CCA]:
[propDict[primaryVar]]}}
pause=0
# In[29]:
# In order to use the same methods I have previously used for examining the data set, go ahead and set this up for the full
data set
# It is not useful in the sense that I already know which values are feasible based on which values are in
criticalVariableList
# It is useful because it allows me the same methods I have used for accessing data in the criticalVariableList to be used
again
def fullPopulateTestedValuesDictionaryUnique(seedName, primaryVar):
 global testedValuesDf
 for seed in fullCriticalVariableList:
     for bscell in fullCriticalVariableList[seed]:
         for cca in fullCriticalVariableList[seed][bscell]:
             if isinstance(fullCriticalVariableList[seed][bscell][cca][0], dict) == False:
                 fullCriticalVariableList[seed][bscell][cca] = pd.Series(fullCriticalVariableList[seed][bscell][cca]).unique()
             tmpAry = fullCriticalVariableList[seed][bscell][cca]
             aryObj = []
             for val in tmpAry:
                 aryObj.append({'value': val, 'tested': False})
             # store the ary of value and weather the value has been tested back to the bscell
             fullCriticalVariableList[seed][bscell][cca] = aryObj
 pause=0
# In[30]:
def populateLengthDictionaryV2(myCriticalVariableList):
 lengthCounter = {}
 for seed in fullCriticalVariableList:
     for bscell in fullCriticalVariableList[seed]:
         if seed in lengthCounter:
             lengthCounter[seed][bscell] = np.NAN
         else: lengthCounter[seed] = {bscell: np.NAN}
 for seed in myCriticalVariableList:
     for bscell in myCriticalVariableList[seed]:
         for cca in myCriticalVariableList[seed][bscell]:
             if np.isnan(lengthCounter[seed][bscell]):
                 lengthCounter[seed][bscell] = 1
             else:
                 lengthCounter[seed][bscell] += 1

 # lengthDf = pd.DataFrame(lengthCounter)
 return lengthCounter
# In[31]:
# determine the number of failed cca's each seed-bscell has
def populateFailureLengthDictionary(uniqueLengthFullDict, successLengthDictMain):
 lengthCounter = {}
 for seed in uniqueLengthFullDict:
     for bscell in uniqueLengthFullDict[seed]:
         if seed in lengthCounter:
             lengthCounter[seed][bscell] = np.nan
         else: lengthCounter[seed] = {bscell: np.nan}
         if seed in successLengthDictMain:
             if bscell in successLengthDictMain[seed]:
                 lengthCounter[seed][bscell] = uniqueLengthFullDict[seed][bscell] - successLengthDictMain[seed][bscell]
 return lengthCounter
# In[32]:
# 1) Determine the number of values a seed had that are not unique
# 2) Determine the number of values a seed has that are unique
# 3) Determine the success of a seed before  permute
# 4) Determine the number of Failed values a seed had before permute
dfTemp =[]
def populateSeedInformationBeforePermute():
global nonUniqueLengthFullDf,uniqueLengthFullDf,successLengthDfMain,failureLengthDfMain
global uniqueLengthFullDict, successLengthDictMain, failureLengthDict
 # Read entire data set reads the entire data set using a parallel read on multiple databases
 # data is placed into dfFull
#########################
#TODO: Remove for testing only
testingApplication = False
#########################
myTime = time.time()
print ('begin reading entire database')
readEntireDataset()
print("Total run time for populating length dataFrame which describes the number of successful values each seed has
before permute --- %s seconds ---" % (time.time() - myTime))
seedList = dfFull.seed.unique()
#####################################################
print("Start run time for populating non unique critical variable value list dictionary")
dfTemp_time = time.time()
#non parallel version
if testingApplication:
  # Approx 100 secs using one db per seed
  populateFullValueList(seedList[4])
else:
  for seed in seedList:
    populateFullValueList(seed)
print("Total run time for populating non unique critical variable value list dictionary --- %s seconds ---" % (time.time() -
dfTemp_time))
####################################
#1) Determine the number of values a seed had that are non unique
dfTemp =[]
 # populate a dictionary that holds the number of non unique values in the data set
myTime = time.time()
#dfTemp = populateLengthDictionary(seedList[0])
#####################################################
print("Start run time for populating non unique length dictionary")
dfTemp_time = time.time()
###parallel varsion not working for some reason #######
#localPool = multiprocessing.Pool(len(seedList))
#for result in tqdm(localPool.imap_unordered(populateLengthDictionary, seedList)):
#    dfTemp.append(result)
#dfTemp = localPool.imap_unordered(populateLengthDictionary, seedList)
#localPool.close()
#localPool.terminate()
#localPool.join()
##########################
#non parallel version
if testingApplication:
  dfTemp.append(pd.DataFrame(populateLengthDictionary(seedList[0])))#, dfFull))
else:
  for seed in seedList:
    dfTemp.append(pd.DataFrame(populateLengthDictionary(seed)))#, dfFull))
print("Total run time for populating length dictionary --- %s seconds ---" % (time.time() - dfTemp_time))
####################################
nonUniqueLengthFullDf = pd.concat(dfTemp, axis=1)
#nonUniqueLengthFullDf = dfTemp[0]
#for index in range(1,len(dfTemp)):
#    nonUniqueLengthFullDf = pd.concat([nonUniqueLengthFullDf, dfTemp[index]], axis=0)
print("Total run time for populating non unique length dataFrame using the full database --- %s seconds ---" % (time.time()
- myTime))
# Just because I was curious if there was a performance difference between these two methods for calculating the
number of values
# in the dataset
# 2) Determine the number of values a seed has that are unique
myTime = time.time()
 # unique Length full df is used for determining the number of unique values for each df
uniqueLengthFullDict = populateLengthDictionaryV2(fullCriticalVariableList)
uniqueLengthFullDf = pd.DataFrame(uniqueLengthFullDict)
print("Total run time for populating length dataFrame using the pre-configured critical vairable list --- %s seconds ---" %
(time.time() - myTime))
 # 3) Determine the success of a seed before permute
```

```
myTime = time.time()
 # get the number of successful values for a bscell
successLengthDictMain = populateLengthDictionaryV2(criticalVariableValueList)
 # if there is no value, then there were no successes
for seed in successLengthDictMain:
    for bscell in successLengthDictMain[seed]:
        if np.isnan(successLengthDictMain[seed][bscell]):
            successLengthDictMain[seed][bscell] = 0

successLengthDfMain = pd.DataFrame(successLengthDictMain)
print("Total run time for populating length dataFrame which describes the number of successful values each seed has
before permute --- %s seconds ---" % (time.time() - myTime))
# 4) Determine the number of Failed values a seed had before permute
myTime = time.time()
#for every row
failureLengthDict = populateFailureLengthDictionary(uniqueLengthFullDict, successLengthDictMain)
 # if there is no value then every cca failed
for seed in failureLengthDict:
    for bscell in failureLengthDict[seed]:
        if np.isnan(failureLengthDict[seed][bscell]):
            failureLengthDict[seed][bscell] = uniqueLengthFullDict[seed][bscell]

# unique Length full df is used for determining the number of unique values for each df
failureLengthDfMain = pd.DataFrame(failureLengthDict)
print("Total run time for populating failure dataFrame which describes the number of failed values each seed has before
permute --- %s seconds ---" % (time.time() - myTime))
# In[33]:
##################################################################################
########## Begin Genetic Algorithm Testing ########################################
def populateFinalDictGeneticAlgorithm():
 global dict, item
 for dict in dataDictGeneticAlgorithm:
     if dict in finalDictGeneticAlgorithm:
         finalDictGeneticAlgorithm[dict][0] += dataDictGeneticAlgorithm[dict][0]
         finalDictGeneticAlgorithm[dict][1] += dataDictGeneticAlgorithm[dict][1]
         finalDictGeneticAlgorithm[dict][2] += dataDictGeneticAlgorithm[dict][2]
         finalDictGeneticAlgorithm[dict][3] += dataDictGeneticAlgorithm[dict][3]
         finalDictGeneticAlgorithm[dict][4] += dataDictGeneticAlgorithm[dict][4]
         finalDictGeneticAlgorithm[dict][5] += dataDictGeneticAlgorithm[dict][5]
         finalDictGeneticAlgorithm[dict][6] += dataDictGeneticAlgorithm[dict][6]
         finalDictGeneticAlgorithm[dict][7] += dataDictGeneticAlgorithm[dict][7]
         finalDictGeneticAlgorithm[dict][8] += dataDictGeneticAlgorithm[dict][8]
         finalDictGeneticAlgorithm[dict][9] += dataDictGeneticAlgorithm[dict][9]
         finalDictGeneticAlgorithm[dict][10] += dataDictGeneticAlgorithm[dict][10]
         finalDictGeneticAlgorithm[dict][11] += dataDictGeneticAlgorithm[dict][11]
         finalDictGeneticAlgorithm[dict][12] += dataDictGeneticAlgorithm[dict][12]
         finalDictGeneticAlgorithm[dict][13] += dataDictGeneticAlgorithm[dict][13]
         finalDictGeneticAlgorithm[dict][14] = dataDictGeneticAlgorithm[dict][14]
         finalDictGeneticAlgorithm[dict][15] = dataDictGeneticAlgorithm[dict][15]
         finalDictGeneticAlgorithm[dict][16] = dataDictGeneticAlgorithm[dict][16]
         finalDictGeneticAlgorithm[dict][17] = dataDictGeneticAlgorithm[dict][17]
     else:
         finalDictGeneticAlgorithm[dict] = [dataDictGeneticAlgorithm[dict][0], dataDictGeneticAlgorithm[dict][1],
dataDictGeneticAlgorithm[dict][2],
                            dataDictGeneticAlgorithm[dict][3], dataDictGeneticAlgorithm[dict][4],
                            dataDictGeneticAlgorithm[dict][5],dataDictGeneticAlgorithm[dict][6], dataDictGeneticAlgorithm[dict][7],
                            dataDictGeneticAlgorithm[dict][8], dataDictGeneticAlgorithm[dict][9], dataDictGeneticAlgorithm[dict][10],
                            dataDictGeneticAlgorithm[dict][11], dataDictGeneticAlgorithm[dict][12],
dataDictGeneticAlgorithm[dict][13],
                            dataDictGeneticAlgorithm[dict][14], dataDictGeneticAlgorithm[dict][15],
dataDictGeneticAlgorithm[dict][16],
                            dataDictGeneticAlgorithm[dict][17]]
def startGeneticAlgorithmPermutationTesting():
 global dataDictGeneticAlgorithm
 dataDictGeneticAlgorithm = Counter()
  # Get the list of seeds
 seedList = dfMain.seed.unique()
  # For each seed family of designs
 for seedName in seedList:
     #print ("Beginning work on seed family %s using primary variable %s" % ( seedName, primaryVar))
     #configTime = time.time()
     #criticalvariableValueList
     #print ("Process %d configuring data structures: %s seconds " % (RANK, time.time() - configTime))
      # make a list of which values have been tested for each BSCELL
      #populateTestedValuesDictionary(seedName, primaryVar)
     #testValuesDf
      # run the substitution algorithm testing for resilience
     print ("Node " + str(RANK) + " beginning genetic algorithm permutation")
     permuteTime = time.time()
     initRowsAndPermuteForSeedSuccessPercentageUsingGeneticAlgorithm(seedName, primaryVar)
     print ("Process %d time it took for genetic algorithm permutation: %s seconds " % (RANK, time.time() - permuteTime))


def initRowsAndPermuteForSeedSuccessPercentageUsingGeneticAlgorithm(seedName, primaryVar):
 global row, dictrows, propDict, permute, saveprop, randomPermute, workQueue, permutePerSeedGeneticAlgorithm,
dataDictGeneticAlgorithm
  # total number of rows that were tested by this process and the number of values left untested of the known
  #  possible feasible values
 totalNumUntested = 0
 totalNumtested = 0
 totalNumberOfValuesTestedForSeed = 0
 myDf = dfMain.loc[dfMain[CONST.seed] == seedName]
 curRowProgress = 0
 totalRows = len(myDf)
 permutePerSeedGeneticAlgorithm = {}
 columnHeadings = list(dfMain)
 startIndex = 0
 endIndex = totalRows
  # start on the proper row for this process
 curRowProgress = startIndex
 endIndex = startIndex + ((endIndex-startIndex) * percentOfRowsToTest)
 if( endIndex < 1):
     endIndex = 1
 # Create a thread for every row. the thread will handle permute for that row and return stats for the row.
 threadList = []
 rowList = []
 threadCount = startIndex
 for curRow in myDf.iloc[int(startIndex):int(endIndex)].values:
     curRow = np.append(curRow,str(threadCount))
     rowList.append(curRow)
     threadCount += 1
 print ("Computing permute with maximum power")
 print("Expect the rows to print in the order they are processed.")
 dataList = []
 #############################################
 # non parallel way
 for row in rowList:
     dataList.append(getRowDataGeneticAlgorithm(row))
 #############################################
 ####Paralle way
 #
 #pool = multiprocessing.Pool(multiprocessing.cpu_count())
 #dataList = pool.map(getRowDataGeneticAlgorithm(), rowList)
 #pool.close()
 #pool.terminate()
 #pool.join()
 #############################################
 '''
 dataDictGeneticAlgorithm[0] numTimes_Cca_Found = float(count[0])
 dataDictGeneticAlgorithm[1]  self.NUM_TIMES_FAILED_PERMUTE = NUM_TIMES_FAILED_PERMUTE
 dataDictGeneticAlgorithm[2]  feasible_after_permute = float(count[2])
 dataDictGeneticAlgorithm[14] NUM_POSSIBLE_VALUES = float(count[15])
 dataDictGeneticAlgorithm[15] Total number of mutations
 dataDictGeneticAlgorithm[16] Total number of passed mutations
 dataDictGeneticAlgorithm[17] Total number of failed mutations
 '''
 print("Finished permute with genetic algorithm")
 for item in dataList:
     seed = item['seed']
     bscell = item['bscell']
     cca = item['cca']
     if (seed, bscell,cca) in dataDictGeneticAlgorithm:
         dataDictGeneticAlgorithm[seed,bscell,cca][0] += item['myDataDict'][seed,bscell,cca][0]
         dataDictGeneticAlgorithm[seed, bscell, cca][1] += item['myDataDict'][seed, bscell, cca][1]
         dataDictGeneticAlgorithm[seed, bscell, cca][2] += item['myDataDict'][seed, bscell, cca][2]
         dataDictGeneticAlgorithm[seed, bscell, cca][14] += item['myDataDict'][seed, bscell, cca][14]
         dataDictGeneticAlgorithm[seed, bscell, cca][15] += item['myDataDict'][seed, bscell, cca][15]
         dataDictGeneticAlgorithm[seed, bscell, cca][16] += item['myDataDict'][seed, bscell, cca][16]
         dataDictGeneticAlgorithm[seed, bscell, cca][17] += item['myDataDict'][seed, bscell, cca][17]
     else : dataDictGeneticAlgorithm[seed,bscell,cca]= item['myDataDict'][seed,bscell,cca]
 print ("Exiting permute with genetic algorithm")
testcounter=0
def getRowDataGeneticAlgorithm(curRow):
 global geneticDictionary
 geneticDictionary = {}
 # create a key value pair of this row's values
 colNum = 0
 rowDict = {}
 columnHeadings = list(dfMain)
 columnHeadings.append("RowID")
 for col in columnHeadings:
     rowDict[col] = curRow[colNum]
     colNum += 1
 myDataDict = {}
 # determine how many values we have
 numTimesToPermute = 0
 localVarValList = criticalVariableValueList
 # figure out how many values we have in this seed the first time we encounter it
 if (rowDict[CONST.seed] in permutePerSeedGeneticAlgorithm):
     numTimesToPermute = permutePerSeedGeneticAlgorithm[rowDict[CONST.seed]]
 else:
     for bscell in localVarValList[rowDict[CONST.seed]]:
         numTimesToPermute += len(localVarValList[rowDict[CONST.seed]][bscell])
     # keep track of how many values were tested for this seed
     permutePerSeedGeneticAlgorithm[rowDict[CONST.seed]] = numTimesToPermute
 # apply the percentage modifier in case the user wants to use less rows than all possible
 numTimesToPermutePercent = int(math.ceil(numTimesToPermute * permRunsPercent))
 # if there was only 1 value then test
 if (numTimesToPermutePercent < 1 and numTimesToPermutePercent > 0):
     numTimesToPermutePercent = 1
 # if you didnt check all the values then go ahead, else all values for this bscell have been tested
 #  numTimesToPermute: number of possible values for this row
 for permute in range(0, numTimesToPermutePercent):
     firstIndexOfUntestedSeed = ""
     firstIndexOfUntestedBscell = ""
     firstIndexOfUntestedCca = ""
     firstIndexOfUntestedCcaVal = -1
     numUntested = 0
     numTested = 0
     firstIndexOfUntested = -1
     # For the very first row, set up the myDataDictionary
     if permute == 0:
         saveprop = rowDict[primaryVar]
         # Using BS_CELL,CCA, seed, count number feasible for each [BS_CELL,CCA,seed] : increment number
         #  found in myDataDict[], using all values from rowDict, See if they have been added to myDataDict yet
         if (rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]) in myDataDict:
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
                 0] += 1  # count 0 -  #numTimesCcaFound
             # everytime this CCA is reencountered, add to the total of values tested
             # myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][15] +=
numTimesToPermute
         else:
             # Initialize this [BS_CELL, CCA,seed] in the myDataDict by adding required info
             # [0=number feasible, 1=number tested, 2=number still feasible after permuation,
             # 3=average cost running total] notify that this node is done working
             # [14] Total values tested
             # [15] Total number of mutations
             # [16] Total number of passed mutations
             # [17] Total number of failed mutations
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]] = [1, 0, 0, 0, 0,
                                                                                            0, 0, 0, 0, 0,
                                                                                            0, 0, 0, 0, 0,
                                                                                            0, 0, 0]
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][3] = str(
                 rowDict[CONST.METRIC_AVERAGE_FOLLOW_END_COST_MOST_LIKELY])
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][6] = str(
                 rowDict[CONST.METRIC_VULNERABILITY])
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][7] = str(
                 rowDict[CONST.DESIGN_SUSTAINED_SPEED])
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][8] = (
                 rowDict[CONST.AAW]).strip(' ')
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][9] = (
                 rowDict[CONST.ASW]).strip(' ')
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][10] = (
                 rowDict[CONST.SUW]).strip(' ')
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][11] = (
                 rowDict[CONST.MIW]).strip(' ')
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][12] = (
                 rowDict[CONST.C2]).strip(' ')
             myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][13] = (
                 rowDict[CONST.IO]).strip(' ')
     # Attempt to randomly get the index of a value in the critical variable list
     chosenSeed = rowDict[CONST.seed]
     chosenbscell = ""
     chosenCCA = ""
     numBscell = len(localVarValList[chosenSeed])
     chosenBscellIndex = random.randint(0, numBscell - 1)
     curBscellIndex = 0
     chosenCcaValIndex = 0
     for bscell in localVarValList[chosenSeed]:
         if curBscellIndex == chosenBscellIndex:
             chosenbscell = bscell
             numCca = len(localVarValList[chosenSeed][chosenbscell])
             chosenCcaIndex = random.randint(0, numCca - 1)
             curCcaIndex = 0
             for cca in localVarValList[chosenSeed][chosenbscell]:
                 if curCcaIndex == chosenCcaIndex:
                     # chosenCCA = localVarValList[chosenSeed][chosenbscell][cca]
                     chosenCCA = cca
                     numValForCca = len(localVarValList[chosenSeed][chosenbscell][chosenCCA])
                     chosenCcaValIndex = random.randint(0, numValForCca - 1)
                     break
                 else:
                     curCcaIndex += 1
             break
         else:
             curBscellIndex += 1
         # Once we have attempted to randomly choose a value to try for this row, make sure we have a
         # random index of a value that hasn't been tested so that we can meet our percentage.
         # If we don't get a random number that hasn't been tested, take the next number that hasn't been tested
     if (localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['tested'] == True):
         seed = rowDict[CONST.seed]
         for bscell in localVarValList[rowDict[CONST.seed]]:
             for cca in localVarValList[seed][bscell]:
                 for curCCAVal in range(0, len(localVarValList[seed][bscell][cca])):
                     if (localVarValList[seed][bscell][cca][curCCAVal]['tested'] == False):
                         if firstIndexOfUntestedCcaVal == -1:
                             # store a reference into the structure to the first untested value
                             firstIndexOfUntestedSeed = seed
                             firstIndexOfUntestedBscell = bscell
                             firstIndexOfUntestedCca = cca
                             firstIndexOfUntestedCcaVal = curCCAVal
                         numUntested += 1
                     else:
                         numTested += 1
         # How often do we state our progress
         '''
         if (permute % (int(math.ceil(numTimesToPermutePercent)) * .10) == 0):
             if (numUntested == 0):
                 # print("No values left to test for this row. Adding to counter number of times a valid value has appeared")
                 a = 0
             else:
                 print( "Untested values for bscell %s in Seed %s: %d. Tested Values for this bscell: %d. Process %d" % (
                         rowDict[CONST.BSCELL], rowDict[CONST.seed], numUntested, numTested, RANK))
                 #print(
                 #"Untested values for bscell %s in Seed %s: %d. Tested Values for this bscell: %d. Process %d row %d" % (
                 #    rowDict[CONST.BSCELL], rowDict[CONST.seed], numUntested, numTested, RANK, curRowProgress))
                 # End of print message
         '''
         randomPermute = \
         localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][firstIndexOfUntestedCca][
             firstIndexOfUntestedCcaVal]['value']
         # if we are allowing duplicate tests of the same value
         localVarValList[firstIndexOfUntestedSeed][firstIndexOfUntestedBscell][firstIndexOfUntestedCca][
             firstIndexOfUntestedCcaVal]['tested'] = True
     else:
         randomPermute = localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['value']
         # if we are allowing duplicate tests of the same value
         if allowDuplicatePermutation == False:
             localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['tested'] = True
     '''
     Up to this point, everything has been the same as the initial version of permute. Minus the pre-configuration.
     The concept of genetic algorithm concept of mutation is introduced here. This means that for a certain percentage
     of the time, rather than trying possible values, we try a new value that is a mutation of the target row value and
     the new target value. We will need to add the new value to the possible values. We also need to randomly choose
     a mutation.
     '''
     ifMutated = False
      #randomly mutate the value we test
     myRand = random.randrange(0,100)
     if (myRand + 1)/100 > mutationChance :
          # were not actually going to check the chosen index this pass so reset it to show it hasnt been selected
         localVarValList[chosenSeed][chosenbscell][chosenCCA][chosenCcaValIndex]['tested'] = False
          # mutate the value
         mutatedValue = mutateValue(rowDict[primaryVar], randomPermute)
          # Since we are going to mutate, we need to add the new value to the list of values. So mutate and add it
         if chosenSeed in geneticDictionary:
             if chosenbscell in geneticDictionary[chosenSeed]:
                 if chosenCCA in geneticDictionary[chosenSeed][chosenbscell]:
                     geneticDictionary[chosenSeed][chosenbscell][chosenCCA].append({
                         'passed': False, 'target':rowDict[primaryVar], 'current':rowDict[primaryVar],
                         'mutant': mutatedValue})
                 else:
                     geneticDictionary[chosenSeed][chosenbscell][chosenCCA] = [{
                         'passed': False, 'target':rowDict[primaryVar], 'current':rowDict[primaryVar],
                         'mutant': mutatedValue}]
             else:
                 geneticDictionary[chosenSeed][chosenbscell] = {chosenCCA:[{
                         'passed': False, 'target':rowDict[primaryVar], 'current':rowDict[primaryVar],
                         'mutant': mutatedValue}]}
         else:
             geneticDictionary[chosenSeed] = {chosenbscell:  {chosenCCA: [{
                         'passed': False, 'target':rowDict[primaryVar], 'current':rowDict[primaryVar],
                         'mutant': mutatedValue}]}}
         rowDict[primaryVar] = mutatedValue
         ifMutated = True
          # Add to the number of mutations
         myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][15] += 1
     else: rowDict[primaryVar] = randomPermute
     # run the datarow through the appropriate regression model for this seed
```

```
    if rowDict[CONST.seed] == CONST.i1:
        rowDict = i1.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.i2:
        rowDict = i2.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.ic:
        rowDict = ic.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.m1:
        rowDict = m1.RegEx(rowDict)
    if rowDict[CONST.seed] == CONST.m2:
        rowDict = m2.RegEx(rowDict)
        # run requirements and metrics on the datarow
    rowDict = requirements.RegEx(rowDict)
    rowDict = metricsv31.RegEx(rowDict)
    '''
    0 numTimes_Cca_Found = float(count[0])
    1  self.NUM_TIMES_FAILED_PERMUTE = NUM_TIMES_FAILED_PERMUTE
    2  feasible_after_permute = float(count[2])
    14 NUM_POSSIBLE_VALUES = float(count[15])
    [15] Total number of mutations
    [16] Total number of passed mutations
    [17] Total number of failed mutations
    '''
    # number of possible values this will include mutations in the count
    myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][14] += 1
    # NOTE: For both the case where a row has been tested and not been tested, increment the appropriate values in the
myDataDict
    # If row is feasible after permute then add to still feasible count and to numtested
    if rowDict[CONST.REQ_CUMULATIVE] > -1:
        # This is a really interesting value. If the row fails feasibility test after permutation then it means that it lacks
resilience
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
            2] += 1  # count 2 - # feasible after permute
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][4] += rowDict[
            CONST.METRIC_VULNERABILITY]
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][5] += rowDict[
            CONST.DESIGN_SUSTAINED_SPEED]
        # if we mutated and passed, set the passed attribute to true and increment the passed counter
        if ifMutated:
            indexVal = 0
            for item in geneticDictionary[chosenSeed][chosenbscell][chosenCCA]:
                if item['mutant'] == mutatedValue:
                    geneticDictionary[chosenSeed][chosenbscell][chosenCCA][indexVal]['passed'] = True
                     # Add to the number of passed mutations
                    myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][16] += 1
                    break
                indexVal+=1
    else:
        # This means the row is no longer feasible so only add to number of rows tested
        myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][
            1] += 1  # count 1 - #NUM_TIMES_FAILED_PERMUTE
         # if we muted and failed, add to the number of failed mutations and set the passed attribute of the
         # mutated value to false
        if ifMutated:
            indexVal = 0
            for item in geneticDictionary[chosenSeed][chosenbscell][chosenCCA]:
                if item['mutant'] == mutatedValue:
                    geneticDictionary[chosenSeed][chosenbscell][chosenCCA][indexVal]['passed'] = False
                     # Add to the number of failed mutations
                    myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][17] += 1
                    break
                indexVal+=1


        # Another row has completed
 #curRowProgress += 1
 totalNumUntested = 0
 totalNumtested = 0
 seed = rowDict[CONST.seed]
 # Reset the values for each row so that each row has fair access to possible values
 for bscell in localVarValList[rowDict[CONST.seed]]:
    for cca in localVarValList[seed][bscell]:
        for curCCAVal in range(0, len(localVarValList[seed][bscell][cca])):
            # While values are being reset, keep track of how many rows have not been tested and
            # also track the number of values that were tested for this seed
            if (localVarValList[seed][bscell][cca][curCCAVal]['tested'] == False):
                totalNumUntested += 1
            else:
                totalNumtested += 1
            localVarValList[seed][bscell][cca][curCCAVal]['tested'] = False


 if(int(rowDict['RowID']) % 50 == 0):
    print ("Computed seed %s bascel %s cca %s. Using row %s" % (rowDict[CONST.seed],rowDict[CONST.BSCELL],
rowDict[CONST.CCA],rowDict['RowID']))
 #for item in myDataDict:
 #    dataDict[item] = myDataDict[item]
 return {'seed':rowDict[CONST.seed],'bscell': rowDict[CONST.BSCELL],'cca': rowDict[CONST.CCA],'myDataDict':
myDataDict,
        'totalNumUntested': totalNumUntested, 'totalNumTested': totalNumtested,
        'totalNumMutations': myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL], rowDict[CONST.CCA]][15],
        'totalNumMutationsPassed': myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL],
rowDict[CONST.CCA]][16],
        'totalNumMutationsFailed': myDataDict[rowDict[CONST.seed], rowDict[CONST.BSCELL],
rowDict[CONST.CCA]][17]}
'''
Mutation possible answers
 0) Somewhere in between
    a) Random value closer to current value
    b) Random value closer to target value
    c) Halfway
    Note: If the new value is already present, then move on without mutation
 1) Above or below current value by whichever puts the point:
    a) Places the current value in between itself and the target value ( new <---- current ------- target )
    b) Places the target value in between itself and the current value ( current ----- target -----> new )
'''
def mutateValue(current, target):
 # choice 0 or 1
  # 0) Somewhere in between
  # 1) Above or below current value by whichever puts the point:
 position = [0, 1]
 positionChoice = random.choice(position)
 # in between choice
  # 0) Random value closer to current value
  # 1) Random value closer to target value
  # 2) Halfway
 inBetween = [0, 1, 2]
 inBetweenChoice = random.choice(inBetween)
 # aboveBelowChoice
  # 0) Places the current value in between itself and the target value ( new <---- current ------- target )
  # 1) Places the target value in between itself and the current value ( current ----- target -----> new )
 aboveBelow = [0, 1]
 aboveBelowChoice = random.choice(aboveBelow)
 # only dealing with positive numbers
 mutatedValue = 0
 halfway = (current+target)/2
 current = int(current)
 halfway = int(halfway)
 target = int(target)
  # force a mutated range
 if halfway == target or halfway == current:
    current = random.randrange(600,1000)
    target = random.randrange(0,400)
    halfway = 500
 current = int(current)
 halfway = int(halfway)
 target = int(target)
```

```
#NOTE: random.uniform(current, halfway) will select floating point number but it may not be a meaningful step
 # Somewhere in between
if positionChoice == 0:
     # Random value closer to current value
    if inBetweenChoice == 0:
        if(current < halfway):
            mutatedValue = random.randrange(current, halfway,1)
        else:
            mutatedValue = random.randrange(halfway, current,1)
     # Random value closer to target value
    elif inBetweenChoice == 1:
        if(target < halfway):
            mutatedValue = random.randrange(target, halfway,1)
        else:
            mutatedValue = random.randrange(halfway, target,1)
     # Halfway
    elif inBetweenChoice == 2:
        mutatedValue = halfway
else: #Above or below current value by whichever puts the point
     # Places the current value in between itself and the target value ( new <---- current ------- target )
    if aboveBelowChoice == 0:
         # ( new <---- current ------- target )
        if(current < halfway):
            mutatedValue = halfway - random.randrange(current, halfway, 1)
        else: # ( target ----- current -------> new )
            mutatedValue = halfway + random.randrange(halfway, current, 1)
     # Places the target value in between itself and the current value ( current ----- target -----> new )
    elif aboveBelowChoice == 1:
        if(target < halfway):
             # ( new <---- target ----- current )
            mutatedValue = halfway - random.randrange(target, halfway,1)
        else:
             # ( current ----- target -----> new )
            mutatedValue = halfway + random.randrange(halfway, target,1)
return mutatedValue
###########END GENETIC ALGORITHM TESTING###############
#########################################################
### start Execution by calling the first major function and its helper functions
# method 1, permutation with substitution
if __name__ == '__main__':
 start_time = time.time()
 mpiCreateDictionaryAndPermute()
 print("Total run time for permute --- %s seconds ---" % (time.time() - start_time))
 # Have the Root process of the mpi run to collect the data from all of the processes
 # and combine that data into one location for determining feasibility for each of
 # the combat capability alternatives
 print( "Process %d: has completed and is passing off data to main" % (RANK))
 #print(criticalVariableValueList['m2']['3B'])
 test = mpi_comm.gather(criticalVariableValueList, root=0)
 if RANK == ROOT:
    print ("root is counting results")
    # make collection containing results from all nodes
    finalDict = Counter()
    populateFinalDict()
    statData = []
    numSeeds = CONST.NUMSEEDS
    hundredMillion = CONST.HUNDREDMILLION
    # After the easier to use dictionary is created in populateFinaDict() from the data then add metrics for amount it
    # costs per hundred million, general upgrade metric, and a metric for the cost per hundred million for ships after
    # the first ship the first few ships after the first ship always cost less. Also the percent of ships that are
    # feasible and the percent of ships that are feasible versus the number tested
    addMetricsAndFeasibilityToDataRows()
    # Now that we have added some more fields to the rows of data using 'addMetricsAndFeasibilityToDataRows()',
    # add  the overall rank of each row to data set. This will identify the single best ship design. This is good except
    # the the single best design may not be the most resilient and it is possible that none of the other designs close
    # to the best design will be feasible. Sort the data rows based on the best overall row of data in the data set.
    addOverallRankAndSortData()
    # Once the best overall row has been found using 'addOverallRankAndSortData()', lets look for the best row of data
    # in each of the CCAs. Since we have the highest overall rank already assigned to each row of data,
    # simply sorting each CCA based on the overall rank will put the CCA rows of data in order
    addCCARankAndSortData()
    # Using 'addCCARankAndSortData()' we added the CCA Rank and sorted the data based on that rank.
    # Now that each row of data has a rank based on its overall performance against all other rows of data
    # and each row also has a rank based on its rank within its own CCA, it
    # is time to determine a the ranking for each of the 5 primary propulsion system configurations ( the 5 seeds )
    addSeedRankAndSortData()
    # At this point each row of data has a overall rank, a CCA rank, and a relative to see rank.
    # Now its time to write the data out to a csv file
    writeDataToCSVFile()
    ##############################################
    ####### Begin Genetic algorithm ############
    # Alot of this is very similiar to the initial permute but we are testing genetic algorithm mutation
    startGeneticAlgorithmPermutationTesting()
    #populate the same information as we did in th regular permute in the Genetic algorithm
    finalDictGeneticAlgorithm = Counter ()
     # I dont know if I need to do this
    populateFinalDictGeneticAlgorithm()
    testDF = pd.DataFrame(finalDictGeneticAlgorithm)
    testDF2  = pd.DataFrame(dataDictGeneticAlgorithm)
    # close the connections
    for x in range(0, numDBs-1):
        if(con[x]):
            con[x].close()
                                                            #Genetic algorithm for permutation selection
```

REFERENCES

A, Z. M. M. n.d. “Extending EXPRESS for Imprecise and Uncertain Engineering Information Modeling.” *Journal of Intelligent Manufacturing* 17 (1): 57–83. doi:10.1007/s10845-005-5513-1.

Achille, Messac. 1996. “Physical Programming: Effective Optimization for Computationl Design.” *AIAA Journal* 34 (1): 149–58.

Akao, Yoji. 2004. *Quality Function Deployment: Integrating Customer Requirements Into Product Design*. Taylor & Francis.

Alexander, Joyce M. 1989. “An Analysis of Conflict in Northern Ireland.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 225–41. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_15.

Antonsson, E. K., and K. N. Otto. 1995. “Imprecision in Engineering Design.” *Journal of Mechanical Design* 117 (B): 25–32. doi:10.1115/1.2836465.

AP, Paul Farley /. 2009. “Unbalanced Warfare Not New for U.S. Navy.” *Msnbc.com*. April 10. http://www.nbcnews.com/id/30157793/ns/us_news-military /t/unbalanced-warfare-not-new-us-navy/.

Art B. Owen. 2017. “Orthogonal Arrays for Computer Experiments, Integration and Visualization.” Accessed April 13. http://www3.stat.sinica.edu.tw/statistica/oldpdf/A2n27.pdf.

Avigad, Gideon, and Jürgen Branke. 2008. “Embedded Evolutionary Multi-Objective Optimization for Worst Case Robustness.” In *Proceedings of the 10th Annual*

Conference on Genetic and Evolutionary Computation, 617–624. GECCO ’08. New York, NY, USA: ACM. doi:10.1145/1389095.1389221.

Barker, Thomas B., and Andrew Milivojevich. 2016. *Quality by Experimental Design, Fourth Edition*. CRC Press.

Barlow, Richard E., C. A. Claroti, and Fabio Spizzichino. 1993. *Reliability and Decision Making*. CRC Press.

Barrico, C., and C. H. Antunes. 2006. “Robustness Analysis in Multi-Objective Optimization Using a Degree of Robustness Concept.” In *2006 IEEE International Conference on Evolutionary Computation*, 1887–92. doi:10.1109/CEC.2006.1688537.

Barrico, Carlos, and Carlos Henggeler Antunes. 2007. “An Evolutionary Approach for Assessing the Degree of Robustness of Solutions to Multi-Objective Models.” In *Evolutionary Computation in Dynamic and Uncertain Environments*, edited by Dr Shengxiang Yang, Dr Yew-Soon Ong, and Dr Yaochu Jin, 565–82. Studies in Computational Intelligence 51. Springer Berlin Heidelberg. doi:10.1007/978-3-540-49774-5_25.

Bennett, James G., and Thomas Lamb. 1995. “The National Shipbuilding Research Program. 1995 Ship Production Symposium. Paper No. 23: Concurrent Engineering Application and Implementation for US Shipbuilding.” DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix =html&identifier=ADA452933.

Ben-Tal, A., and A. Nemirovski. 1998. “Robust Convex Optimization.” *Mathematics of Operations Research* 23 (4): 769–805. doi:10.1287/moor.23.4.769.

Bernstein, Joshua I. (Joshua Ian). 1998. “Design Methods in the Aerospace Industry : Looking for Evidence of Set-Based Practices.” Thesis, Massachusetts Institute of Technology. http://dspace.mit.edu/handle/1721.1/82675.

Bogen, A. Christopher, Mahbubur Rashid, and others. 2013. “Evaluating Data Clustering Approach for Life-Cycle Facility Control.” DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA578625.

Box, George E. P., William Gordon Hunter, and J. Stuart Hunter. 1978. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. Wiley Series in Probability and Mathematical Statistics. New York: Wiley.

Bramanti, A., P. Di Barba, M. Farina, and A. Savini. 2001. “Combining Response Surfaces and Evolutionary Strategies for Multiobjective Pareto-Optimization in Electromagnetics.” *International Journal of Applied Electromagnetics and Mechanics* 15 (1–4): 231–36.

Branke, Jürgen. 1998. “Creating Robust Solutions by Means of Evolutionary Algorithms.” In *Parallel Problem Solving from Nature — PPSN V*, edited by Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, 119–28. Lecture Notes in Computer Science 1498. Springer Berlin Heidelberg. doi:10.1007/BFb0056855.

Cappelleri, David J., Mary I. Frecker, Timothy W. Simpson, and Alan Snyder. 2002. “Design of a PZT Bimorph Actuator Using a Metamodel-Based Approach.” *Journal of Mechanical Design* 124 (2): 354. doi:10.1115/1.1446866.

Chandra, Fiona, Dennice F. Gayme, Lijun Chen, and John C. Doyle. 2011. "Robustness, Optimization, and Architectures." *European Journal of Control* 17 (5–6): 472–82. doi:10.3166/ejc.17.472-482.

Che, Jianguo, Jing Wang, and Kai Li. 2014. "A Monte Carlo Based Robustness Optimization Method in New Product Design Process: A Case Study." *American Journal of Industrial and Business Management* 4 (7): 360–69. doi:10.4236/ajibm.2014.47044.

Chen, Shu-Jen, and Ching-Lai Hwang. 1992a. "Fuzzy Multiple Attribute Decision Making Methods." In *Fuzzy Multiple Attribute Decision Making*, 289–486. Lecture Notes in Economics and Mathematical Systems 375. Springer Berlin Heidelberg. doi:10.1007/978-3-642-46768-4_5.

———. 1992b. "Fuzzy Ranking Methods." In *Fuzzy Multiple Attribute Decision Making*, 101–288. Lecture Notes in Economics and Mathematical Systems 375. Springer Berlin Heidelberg. doi:10.1007/978-3-642-46768-4_4.

———. 1992c. "Fuzzy Sets and Their Operations." In *Fuzzy Multiple Attribute Decision Making*, 42–100. Lecture Notes in Economics and Mathematical Systems 375. Springer Berlin Heidelberg. doi:10.1007/978-3-642-46768-4_3.

———. 1992d. "Multiple Attribute Decision Making — An Overview." In *Fuzzy Multiple Attribute Decision Making*, 16–41. Lecture Notes in Economics and Mathematical Systems 375. Springer Berlin Heidelberg. doi:10.1007/978-3-642-46768-4_2.

Chen, Wei, Jan Unkelbach, Alexei Trofimov, Thomas Madden, Hanne Kooy, Thomas Bortfeld, and David Craft. 2012. "Including Robustness in Multi-Criteria

Optimization for Intensity-Modulated Proton Therapy." *Physics in Medicine and Biology* 57 (3): 591–608. doi:10.1088/0031-9155/57/3/591.

Chen, Yen-Sen. 2016. "Compressible and Incompressible Flow Computations with a Pressure Based Method." In *27th Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics. Accessed September 1. http://arc.aiaa.org /doi/abs/10.2514/6.1989-286.

Chen, Yen-Sen, and Richard Farmer. 2016. "CFD Analysis of Baffle Flame Stabilization." In *27th Joint Propulsion Conference*. American Institute of Aeronautics and Astronautics. Accessed September 1. http://arc.aiaa.org /doi/abs/10.2514/6.1991-1967.

Correa Florez, Carlos A., Ricardo A. Bolaños Ocampo, and Antonio H. Escobar Zuluaga. 2014. "Multi-Objective Transmission Expansion Planning Considering Multiple Generation Scenarios." *International Journal of Electrical Power & Energy Systems* 62 (November): 398–409. doi:10.1016/j.ijepes.2014.04.063.

Daum, D. A., K. Deb, and J. Branke. 2007. "Reliability-Based Optimization for Multiple Constraints with Evolutionary Algorithms." In *2007 IEEE Congress on Evolutionary Computation*, 911–18. doi:10.1109/CEC.2007.4424567.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan. 2002. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *Trans. Evol. Comp* 6 (2): 182–197. doi:10.1109/4235.996017.

Deb, Kalyanmoy, Shamik Chaudhuri, and Kaisa Miettinen. 2006. "Towards Estimating Nadir Objective Vector Using Evolutionary Approaches." In *Proceedings of the*

8th Annual Conference on Genetic and Evolutionary Computation, 643–650. ACM. http://dl.acm.org/citation.cfm?id=1144113.

Deb, Kalyanmoy, and Tushar Goel. 2001a. "A Hybrid Multi-Objective Evolutionary Approach to Engineering Shape Design." In *Evolutionary Multi-Criterion Optimization*, edited by Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne, 385–99. Lecture Notes in Computer Science 1993. Springer Berlin Heidelberg. doi:10.1007/3-540-44719-9_27.

———. 2001b. "Controlled Elitist Non-Dominated Sorting Genetic Algorithms for Better Convergence." In *Evolutionary Multi-Criterion Optimization*, edited by Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne, 67–81. Lecture Notes in Computer Science 1993. Springer Berlin Heidelberg. doi:10.1007/3-540-44719-9_5.

———. 2003. "Multi-Objective Evolutionary Algorithms for Engineering Shape Design." In *Evolutionary Optimization*, 147–75. International Series in Operations Research & Management Science 48. Springer US. doi:10.1007/0-306-48041-7_6.

Deb, Kalyanmoy, and Himanshu Gupta. 2005. "Searching for Robust Pareto-Optimal Solutions in Multi-Objective Optimization." In *Evolutionary Multi-Criterion Optimization*, edited by Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, 150–64. Lecture Notes in Computer Science 3410. Springer Berlin Heidelberg. doi:10.1007/978-3-540-31880-4_11.

———. 2006. "Introducing Robustness in Multi-Objective Optimization." *Evolutionary Computation* 14 (4): 463–494.

Deb, Kalyanmoy, Kaisa Miettinen, and Deepak Sharma. 2009. “A Hybrid Integrated Multi-Objective Optimization Procedure for Estimating Nadir Point.” In *International Conference on Evolutionary Multi-Criterion Optimization*, 569–583. Springer. http://link.springer.com/chapter/10.1007/978-3-642-01020-0_44.

Deb, Kalyanmoy, Amrit Pratap, and Subrajyoti Moitra. 2000. “Mechanical Component Design for Multiple Ojectives Using Elitist Non-Dominated Sorting GA.” In *Parallel Problem Solving from Nature PPSN VI*, edited by Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, 859–68. Lecture Notes in Computer Science 1917. Springer Berlin Heidelberg. doi:10.1007/3-540-45356-3_84.

Dhingra, A. K., and Singiresu S. Rao. 1989. “Integrated Optimal Design of Planar Mechanisms Using Fuzzy Theories.” In . https://miami.pure.elsevier.com/en/publications/integrated-optimal-design-of-planar-mechanisms-using-fuzzy-theori.

Dhingra, Anoop K., Singiresu S. Rao, and Virendra Kumar. 1992. “Nonlinear Membership Functions in Multiobjective Fuzzy Optimization of Mechanical and Structural Systems.” *AIAA Journal* 30 (1): 251–60. doi:10.2514/3.10906.

Diaz, A. R. 1989. “A Strategy for Optimal Design of Hierarchical Systems Using Fuzzy Sets.” In *The 1989 NSF Engineering Design Research Conference*, 537–547.

Diaz, Alejandro R. 1988. “Fuzzy Set Based Models in Design Optimization.” In . https://scholars.opb.msu.edu/en/publications/fuzzy-set-based-models-in-design-optimization-3.

Ditlevsen, Ove. 1979. “Narrow Reliability Bounds for Structural Systems.” *Journal of Structural Mechanics* 7 (4): 453–72. doi:10.1080/03601217908905329.

Doerry, Norbert, and Howard Fireman. 2009. “Fleet Capabilities-Based Assessment.” *Naval Engineers Journal* 121 (4): 107–116.

Dolan, James G. 1989. “Choosing Initial Antibiotic Therapy for Acute Pyelonephritis.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 213–24. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_14.

Doolittle, Erin K., Hervé LM Kerivin, and Margaret M. Wiecek. 2012. “A Robust Multiobjective Optimization Problem with Application to Internet Routing.” In *Tech. Rep. R2012-11-DKW, Clemson University*. https://www.clemson.edu/science/departments/mathematical-sciences/documents/technical-reports/TR2015-11-ed.hk.mw.pdf.

Dornberger, Rolf, and Dirk Bche. 2002. “Multidisciplinary Optimization In Turbomachinery Design.” *ResearchGate*, August. https://www.researchgate.net /publication/2527400_Multidisciplinary_Optimization_In_Turbomachinery_Design.

Dubois, Didier. 1991. “Fuzzy Sets and Their Applications : Vilem Novak, Translated from Czechoslovakian. Bristol and Philadelphia: Adam Hilger, 1989, 248 Pages.” *Mathematical Social Sciences* 21 (2): 193–97.

Dubois, Didier J. 1980. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press.

Efstathiou, J., and V. Rajkovic. 1979. "Multiattribute Decisionmaking Using a Fuzzy Heuristic Approach." *IEEE Transactions on Systems, Man, and Cybernetics* 9 (6): 326–33. doi:10.1109/TSMC.1979.4310221.

Ehrgott, Matthias, Jonas Ide, and Anita Schöbel. 2014. "Minmax Robustness for Multi-Objective Optimization Problems." *European Journal of Operational Research* 239 (1): 17–31. doi:10.1016/j.ejor.2014.03.013.

Emmerich, M. T. M., K. C. Giannakoglou, and B. Naujoks. 2006. "Single- and Multiobjective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels." *IEEE Transactions on Evolutionary Computation* 10 (4): 421–39. doi:10.1109/TEVC.2005.859463.

Emmerich, Michael, Alexios Giotis, Mutlu Özdemir, Thomas Bäck, and Kyriakos Giannakoglou. 2002. "Metamodel—Assisted Evolution Strategies." In *Parallel Problem Solving from Nature — PPSN VII*, edited by Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, Hans-Paul Schwefel, and José-Luis Fernández-Villacañas, 361–70. Lecture Notes in Computer Science 2439. Springer Berlin Heidelberg. doi:10.1007/3-540-45712-7_35.

Eskandarpour, Majid, Ehsan Nikbakhsh, and Seyed Hessameddin Zegordi. 2014. "Variable Neighborhood Search for the Bi-Objective Post-Sales Network Design Problem: A Fitness Landscape Analysis Approach." *Computers & Operations Research*, Recent advances in Variable neighborhood search, 52, Part B (December): 300–314. doi:10.1016/j.cor.2013.06.002.

Evans, J. Harvey. 1959. "Basic Design Concepts." *Journal of the American Society for Naval Engineers* 71 (4): 671–78. doi:10.1111/j.1559-3584.1959.tb01836.x.

Farina, M., A. Bramanti, and P. Di Barba. 2002. “A GRS Method for Pareto-Optimal Front Identification in Electromagnetic Multiobjective Synthesis.” In *Computation in Electromagnetics, 2002. CEM 2002. The Fourth International Conference on (Ref. No. 2002/063),* 2 pp.-. doi:10.1049/ic:20020161.

Farina, M., and J. K. Sykulski. 2001. “Comparative Study of Evolution Strategies Combined with Approximation Techniques for Practical Electromagnetic Optimization Problems.” *IEEE Transactions on Magnetics* 37 (5): 3216–20. doi:10.1109/20.952580.

Fatti, L. Paul. 1989. “Water Research Planning in South Africa.” In *The Analytic Hierarchy Process,* edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 122–37. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_8.

Fireman, Howard, Marianne Nutting, Tom Rivers, Gary Carlile, and Kendall King. 1998. “LPD 17 on the Shipbuilding Frontier: Integrated Product & Process Development.” In *Proceedings of the Association of Scientists and Engineers 35th Annual Technical Symposium*. http://fas.org/man/dod-101/sys/ship/docs/ase98sbf.pdf.

Fishburn, Peter C. 1982a. “Expected Utility for Probability Measures.” In *The Foundations of Expected Utility,* 23–34. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_3.

———. 1982b. “Lexicographic Quasilinear Utility.” In *The Foundations of Expected Utility,* 35–55. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_4.

———. 1982c. “Linear Utilities on Product Sets.” In *The Foundations of Expected Utility*, 73–84. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_6.

———. 1982d. “Linear Utility for Partially Ordered Preferences.” In *The Foundations of Expected Utility*, 57–71. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_5.

———. 1982e. “Multilinear Utility for Probability Measures.” In *The Foundations of Expected Utility*, 99–103. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_8.

———. 1982f. “Multilinear Utility on Products of Mixture Sets.” In *The Foundations of Expected Utility*, 85–98. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_7.

———. 1982g. “Subjective Expected Utility for Arbitrary State Sets.” In *The Foundations of Expected Utility*, 121–34. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_10.

———. 1982h. “Subjective Linear Utility for Partially Ordered Preferences.” In *The Foundations of Expected Utility*, 135–47. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_11.

———. 1982i. “Subjective Linear Utility on Products of Mixture Sets.” In *The Foundations of Expected Utility*, 107–20. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_9.

———. 1982j. “Subjective Linear Utility with Conditional Preference Comparisons.” In *The Foundations of Expected Utility*, 149–68. Theory and Decision Library 31. Springer Netherlands. doi:10.1007/978-94-017-3329-8_12.

Fliege, Jörg, and Ralf Werner. 2014. “Robust Multiobjective Optimization & Applications in Portfolio Optimization.” *European Journal of Operational Research*, 60 years following Harry Markowitz’s contribution to portfolio theory and operations research, 234 (2): 422–33. doi:10.1016/j.ejor.2013.10.028.

Fonseca, Carlos M., Peter J. Fleming, and others. 1993. “Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization.” In *Icga*, 93:416–423. Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.9077&rep=rep1&type=pdf.

French, Simon. 2016. “Decision Theory: An Introduction to the Mathematics of Rationality (Ellis Horwood Series in Mathematics and Its Applications): Simon French: 9780470203088: Amazon.com: Books.” Accessed September 1. https://www.amazon.com/Decision-Theory-Introduction-Mathematics-Applications/dp/0470203080.

Gabrel, Virginie, Cécile Murat, and Aurélie Thiele. 2014. “Recent Advances in Robust Optimization: An Overview.” *European Journal of Operational Research* 235 (3): 471–83. doi:10.1016/j.ejor.2013.09.036.

Ganesan, Vikram. 2001. “Global Optimization of the Nonconvex Containership Design Problem Using the Reformulation-Linearization Technique.” Virginia Polytechnic Institute and State University. http://theses.lib.vt.edu/theses/available/etd-08142001-202530/.

Garner, Matt, Norbert Doerry, Adrian MacKenna, Frank Pearce, Chris Bassler, Shari Hannapel, and Peter McCauley. 2015. “Concept Exploration Methods for the Small Surface Combatant.” In *World Maritime Technology Conference*, 3–7. http://doerry.org/norbert/papers/20150717ssc-ce.pdf.

Goberna, M. A., V. Jeyakumar, G. Li, and J. Vicente-Pérez. 2015. “Robust Solutions to Multi-Objective Linear Programs with Uncertain Data.” *European Journal of Operational Research* 242 (3): 730–43. doi:10.1016/j.ejor.2014.10.027.

Goel, Tushar, and Kalyanmoy Deb. 2001. “Hybrid Methods for Multi-Objective Evolutionary Algorithms.” In *KANGAL REPORT NUMBER*, 200–1.

Goel, Tushar, Rajkumar Vaidyanathan, Raphael T. Haftka, Wei Shyy, Nestor V. Queipo, and Kevin Tucker. 2007. “Response Surface Approximation of Pareto Optimal Front in Multi-Objective Optimization.” *Computer Methods in Applied Mechanics and Engineering* 196 (4–6): 879–93. doi:10.1016/j.cma.2006.07.010.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Golden, Bruce L., and Qiwen Wang. 1989. “An Alternate Measure of Consistency.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 68–81. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_5.

Golden, Bruce L., Edward A. Wasil, and Patrick T. Harker. 1989. “Introduction.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant

Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 1–2. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_1.

Golden, Bruce L., Edward A. Wasil, and Doug E. Levy. 1989. "Applications of the Analytic Hierarchy Process: A Categorized, Annotated Bibliography." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 37–58. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_3.

Gray, Alexander W., and David J. Singer. 2008. "Impacts of Type-2 Fuzzy Modeling Approach on Set-Based Design, Based on Results from an Existing Hybrid Agent Design Experiment." In . https://experts.umich.edu/en/publications/impacts-of-type-2-fuzzy-modeling-approach-on-set-based-design-bas.

Gray, C. T. 1988. "Introduction to Quality Engineering: Designing Quality into Products and Processes, G. Taguchi, Asian Productivity Organization, 1986. Number of Pages: 191. Price: $29 (U.K.) - Gray - 1988 - Quality and Reliability Engineering International - Wiley Online Library." April. http://onlinelibrary.wiley.com/doi/10.1002/qre.4680040216/abstract.

Gunawan, S., and S. Azarm. 2004. "Multi-Objective Robust Optimization Using a Sensitivity Region Concept." *Structural and Multidisciplinary Optimization* 29 (1): 50–60. doi:10.1007/s00158-004-0450-8.

Hammersley, J. M., and D. C. Handscomb. 1964. *Monte Carlo Methods*. Monographs on Applied Probability and Statistics. Springer Netherlands.

Hannapel, Shari, and Nickolas Vlahopoulos. 2010. “Introducing Uncertainty in Multidiscipline Ship Design.” *Naval Engineers Journal* 122 (2): 41–52. doi:10.1111/j.1559-3584.2010.00267.x.

Harker, Patrick T. 1989. “The Art and Science of Decision Making: The Analytic Hierarchy Process.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 3–36. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_2.

Hauser, John R., and Don Clausing. 1988. “The House of Quality.” *Harvard Business Review*. May 1. https://hbr.org/1988/05/the-house-of-quality.

Hazır, Öncü, Erdal Erel, and Yavuz Günalay. 2011. “Robust Optimization Models for the Discrete Time/Cost Trade-off Problem.” *International Journal of Production Economics* 130 (1): 87–95. doi:10.1016/j.ijpe.2010.11.018.

He, Jim, Shari Hannapel, and Nickolas Vlahopoulos. 2011. “Multidisciplinary Design Optimization of Ship Hull Forms Using Metamodels.” In . doi:10.1115/DETC2011-47761.

“History of the United States Navy.” 2017. *Wikipedia*. https://en.wikipedia.org/w/index.php?title=History_of_the_United_States_Navy&oldid=758454658.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation*. 3 edition. Boston: Pearson.

Hunter, J. Stuart. 1985. “ASQ: Statistical Design Applied to Product Design.” http://asq.org/qic/display-item/?item=5515.

Ide, Jonas, Elisabeth Köbis, Daishi Kuroiwa, Anita Schöbel, and Christiane Tammer. 2014. “The Relationship between Multi-Objective Robustness Concepts and Set-Valued Optimization.” *Fixed Point Theory and Applications* 2014 (1): 83. doi:10.1186/1687-1812-2014-83.

Ide, Jonas, Morten Tiedemann, Stephan Westphal, and Felix Haiduk. 2014. “An Application of Deterministic and Robust Optimization in the Wood Cutting Industry.” *4OR* 13 (1): 35–57. doi:10.1007/s10288-014-0265-4.

Ishibuchi, H., T. Yoshida, and T. Murata. 2003. “Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling.” *IEEE Transactions on Evolutionary Computation* 7 (2): 204–23. doi:10.1109/TEVC.2003.810752.

Iversen, Gudmund R. 1984. *Bayesian Statistical Inference*. SAGE.

Jahn, Johannes, and Truong Xuan Duc Ha. 2011. “New Order Relations in Set Optimization.” *Journal of Optimization Theory and Applications* 148 (2): 209–36. doi:10.1007/s10957-010-9752-8.

Jain, Anil K., and Richard C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice Hall PTR.

Jain, Ramesh. 1976. “Decisionmaking in the Presence of Fuzzy Variables.” *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6 (10): 698–703. doi:10.1109/TSMC.1976.4309421.

John, Peter W. M. 1998. “Front Matter.” In *Statistical Design and Analysis of Experiments*, i–xxiv. Classics in Applied Mathematics. Society for Industrial and

Applied Mathematics. http://epubs.siam.org/doi/abs/10.1137 /1.9781611971149.fm.

Josephson, John R., Balakrishnan Chandrasekaran, Mark Carroll, Naresh Iyer, Bryon Wasacz, Giorgio Rizzoni, Qingyuam Li, and David A. Erb. 1998. "An Architecture for Exploring Large Design Spaces." In *AAAI/IAAI*, 143–150. http://www.aaai.org/Papers/AAAI/1998/AAAI98-020.pdf.

Kackar, Raghu N. 1989. "Off-Line Quality Control, Parameter Design, and the Taguchi Method." In *Quality Control, Robust Design, and the Taguchi Method*, edited by Khosrow Dehnad, 51–76. Springer US. doi:10.1007/978-1-4684-1472-1_4.

Kaitaniemi, Pekka, Annette Scheiner, Tero Klemola, and Kai Ruohomäki. 2012. "Multi-Objective Optimization Shapes Ecological Variation." *Proceedings. Biological Sciences / The Royal Society* 279 (1729): 820–25. doi:10.1098/rspb.2011.1371.

Kang, Eunsuk, Ethan Jackson, and Wolfram Schulte. 2010. "An Approach for Effective Design Space Exploration." In *Monterey Workshop*, 33–54. Springer. http://link.springer.com/10.1007%2F978-3-642-21292-5_3.

Katrin, Witting. 2012. "Numerical Algorithms for the T - Title - Veröffentlichungen Der Universität - Digitale Sammlungen." http://digital.ub.uni-paderborn.de/hs/content/titleinfo/355856.

Kaufmann, Arnold, and Madan M. Gupta. 1988. *Fuzzy Mathematical Models in Engineering and Management Science*. New York, NY, USA: Elsevier Science Inc.

Keane, R. G., and B. F. Tibbitts. 1996. “A REVOLUTION IN WARSHIP DESIGN: NAVY-INDUSTRY INTEGRATED PRODUCT TEAMS.” https://trid.trb.org/view.aspx?id=480852.

Keane, Robert G., John Mcintire, Howard Fireman, and Daniel J. Maher. 2009. “The LPD 17 Ship Design: Leading a Sea Change Toward Collaborative Product Development.” *Naval Engineers Journal* 121 (2): 15–61. doi:10.1111/j.1559-3584.2009.00189.x.

Keeney, Ralph L., and Howard Raiffa. 2016. “Decisions with Multiple Objectives: Preferences and Value Tradeoffs: Ralph L. Keeney, Howard Raiffa: 9780521438834: Amazon.com: Books.” Accessed September 1. https://www.amazon.com/Decisions-Multiple-Objectives-Preferences-Tradeoffs/dp/0521438837.

Kennedy, Graeme J. 2016. “A Full-Space Barrier Method for Stress-Constrained Discrete Material Design Optimization.” *Structural and Multidisciplinary Optimization* 54 (3): 619–39. doi:10.1007/s00158-016-1428-z.

Kim, I. Y., and O. L. de Weck. 2006. “Adaptive Weighted Sum Method for Multiobjective Optimization: A New Method for Pareto Front Generation.” *Structural and Multidisciplinary Optimization* 31 (2): 105–16. doi:10.1007/s00158-005-0557-6.

Klamroth, K., E. Köbis, A. Schöbel, and Chr. Tammer. 2013. “A Unified Approach for Different Concepts of Robustness and Stochastic Programming via Non-Linear Scalarizing Functionals.” *Optimization* 62 (5): 649–71. doi:10.1080/02331934.2013.769104.

Klir, George J., and Tina A. Folger. 1988. *Fuzzy Sets, Uncertainty and Information*. First Edition edition. Englewood Cliffs, N.J: Prentice Hall.

Knowles, J. 2006. “ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems.” *IEEE Transactions on Evolutionary Computation* 10 (1): 50–66. doi:10.1109/TEVC.2005.851274.

Knowles, J., and D. Corne. 1999. “The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation.” In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99*, 1:105 Vol. 1. doi:10.1109/CEC.1999.781913.

Knowles, Joshua D., and David W. Corne. n.d. “Approximating the Nondominated Front Using.” *Evolutionary Computation* 8 (2).

Knowles, Joshua, and Evan J. Hughes. 2005. “Multiobjective Optimization on a Budget of 250 Evaluations.” In *Evolutionary Multi-Criterion Optimization*, edited by Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, 176–90. Lecture Notes in Computer Science 3410. Springer Berlin Heidelberg. doi:10.1007/978-3-540-31880-4_13.

Kuroiwa, Daishi. 2001. “On Set-Valued Optimization.” *Nonlinear Analysis: Theory, Methods & Applications* 47 (2): 1395–1400. doi:10.1016/S0362-546X(01)00274-7.

Kuroiwa, Daishi, and Gue Myung Lee. 2012. “On Robust Multiobjective Optimization.” *Vietnam J. Math* 40 (2–3): 305–317.

Lee, Jongsoo, and Yong Sik Kwon. 2012. "Conservative Multi-Objective Optimization Considering Design Robustness and Tolerance: A Quality Engineering Design Approach." *Structural and Multidisciplinary Optimization* 47 (2): 259–72. doi:10.1007/s00158-012-0823-3.

Lewis, Robert, and Doug E. Levy. 1989. "Predicting a National Acid Rain Policy." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 155–70. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_10.

Liberatore, Matthew J. 1989. "A Decision Support Approach for R&D Project Selection." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 82–100. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_6.

Liker, J. K., D. K. Sobek, A. C. Ward, and J. J. Cristiano. 1996. "Involving Suppliers in Product Development in the United States and Japan: Evidence for Set-Based Concurrent Engineering." *IEEE Transactions on Engineering Management* 43 (2): 165–78. doi:10.1109/17.509982.

Liker, Jeffrey K. 2004. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. 1 edition. New York: McGraw-Hill Education.

Liu, Zhifeng, Sez Atamturktur, and C. Hsein Juang. 2014. "Reliability Based Multi-Objective Robust Design Optimization of Steel Moment Resisting Frame Considering Spatial Variability of Connection Parameters." *Engineering Structures* 76 (October): 393–403. doi:10.1016/j.engstruct.2014.07.024.

Llopis-Albert, Carlos, Francisco Rubio, and Francisco Valero. 2015. “Improving Productivity Using a Multi-Objective Optimization of Robotic Trajectory Planning.” *Journal of Business Research*, Special issue on The Spirit of StrategySpecial issue on The Spirit of Strategy, 68 (7): 1429–31. doi:10.1016/j.jbusres.2015.01.027.

MacCormac, Earl R. 1989. “Forecasting Loads and Designing Rates for Electric Utilities.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 138–54. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_9.

Mack, Yolanda, Tushar Goel, Wei Shyy, and Raphael Haftka. 2007. “Surrogate Model-Based Optimization Framework: A Case Study in Aerospace Design.” In *Evolutionary Computation in Dynamic and Uncertain Environments*, edited by Dr Shengxiang Yang, Dr Yew-Soon Ong, and Dr Yaochu Jin, 323–42. Studies in Computational Intelligence 51. Springer Berlin Heidelberg. doi:10.1007/978-3-540-49774-5_14.

Madsen, Jens I., Wei Shyy, and Raphael T. Haftka. 2000. “Response Surface Techniques for Diffuser Shape Optimization.” *AIAA Journal* 38 (9): 1512–18. doi:10.2514/2.1160.

Mäkinen, Raino A.E., Jacques Periaux, and Jari Toivanen. 1999. “Multidisciplinary Shape Optimization in Aerodynamics and Electromagnetics Using Genetic Algorithms.” *International Journal for Numerical Methods in Fluids* 30 (2): 149–59. doi:10.1002/(SICI)1097-0363(19990530)30:2<149::AID-FLD829>3.0.CO;2-B.

Mavrotas, George, José Rui Figueira, and Eleftherios Siskos. 2015. “Robustness Analysis Methodology for Multi-Objective Combinatorial Optimization Problems and Application to Project Selection.” *Omega* 52 (April): 142–55. doi:10.1016/j.omega.2014.11.005.

Miettinen, Kaisa. 1998. *Nonlinear Multiobjective Optimization*. Vol. 12. International Series in Operations Research & Management Science. Boston, MA: Springer US. http://link.springer.com/10.1007/978-1-4615-5563-6.

Might, Robert J., and William D. Daniel Jr. 1989. “Decision Support for War Games.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 171–81. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_11.

Mitchell, Kenneth H., and Edward A. Wasil. 1989. “AHP in Practice: Applications and Observations from a Management Consulting Perspective.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 192–212. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_13.

Morgan, James M., and Jeffrey K. Liker. 2006. *The Toyota Product Development System: Integrating People, Process And Technology*. 1 edition. New York: Productivity Press.

Muller, K., and M. Tharigen. 1994. “Applications of Fuzzy Hierarchies and Fuzzy MADM Methods to Innovative System Design.” In *, Proceedings of the Third IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence*, 364–67 vol.1. doi:10.1109/FUZZY.1994.343671.

“Multiobjective Decision Making: Theory and Methodology.” 2016. Accessed September 1. http://store.doverpublications.com/0486462897.html.

Myers, Raymond H., Douglas C. Montgomery, and Christine M. Anderson-Cook. 2016. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons.

Nahm, Y.-E., and H. Ishikawa. 2006. “A New 3D-CAD System for Set-Based Parametric Design.” *The International Journal of Advanced Manufacturing Technology* 29 (1–2): 137–50. doi:10.1007/s00170-004-2213-5.

Nain, Pawan K. S., and Kalyanmoy Deb. 2016. “A Multi-Objective Search and Optimization Procedure with Successive Approximate Models.” Accessed September 1. http://www.egr.msu.edu/~kdeb/papers/k2004012.pdf.

Negoiţă, Dr C. V., and D. A. Ralescu. 1975a. “Deciding in Fuzzy Environment.” In *Applications of Fuzzy Sets to Systems Analysis*, 152–68. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_7.

———. 1975b. “Fuzzy Automata, Fuzzy Languages, and Fuzzy Algorithms.” In *Applications of Fuzzy Sets to Systems Analysis*, 122–51. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_6.

———. 1975c. “Fuzzy Clustering.” In *Applications of Fuzzy Sets to Systems Analysis,* 169–79. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_8.

———. 1975d. “Fuzzy Logic.” In *Applications of Fuzzy Sets to Systems Analysis,* 65–84. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_4.

———. 1975e. “Fuzzy Sets, L-Sets, Flou Sets.” In *Applications of Fuzzy Sets to Systems Analysis,* 12–42. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_2.

———. 1975f. “Fuzzy Systems.” In *Applications of Fuzzy Sets to Systems Analysis,* 85–121. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_5.

———. 1975g. “Fuzzy Theories.” In *Applications of Fuzzy Sets to Systems Analysis,* 43–64. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Birkhäuser Basel. doi:10.1007/978-3-0348-5921-9_3.

Nicoglou, Antonine. 2015. “The Evolution of Phenotypic Plasticity: Genealogy of a Debate in Genetics.” *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences* 50 (April): 67–76. doi:10.1016/j.shpsc.2015.01.003.

Obayashi, Shigeru, Daisuke Sasaki, and Akira Oyama. 2004. “Finding Tradeoffs by Using Multiobjective Optimization Algorithms.” *Transactions of the Japan Society for Aeronautical and Space Sciences* 47 (155): 51–58.

Ong, Yew S., Prasanth B. Nair, and Andrew J. Keane. 2003. “Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling.” *AIAA Journal* 41 (4): 687–96. doi:10.2514/2.1999.

O’Rourke, Ronald. 2009. “Navy CG (X) Cruiser Program: Background, Oversight Issues, and Options for Congress.” In . DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA513538.

Osiadacz, Andrzej J. 1989. “Multiple Criteria Optimization; Theory, Computation, and Application, Ralph E. Steuer, Wiley Series in Probability and Mathematical Statistics - Applied, Wiley, 1986, No. of Pages 546, Price f5 1.40, $77.10.” *Optimal Control Applications and Methods* 10 (1): 89–90. doi:10.1002/oca.4660100109.

Otto, E. N., and E. K. Antonsson. 1994. “Modeling Imprecision in Product Design.” In *, Proceedings of the Third IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence*, 346–51 vol.1. doi:10.1109/FUZZY.1994.343674.

Otto, Kevin N., and Erik K. Antonsson. 1991. “Trade-off Strategies in Engineering Design.” *Research in Engineering Design* 3 (2): 87–103.

———. 1994. “Design Parameter Selection in the Presence of Noise.” *Research in Engineering Design* 6 (4): 234–246.

Papila, Nilay, Wei Shyy, Lisa Griffin, and Daniel Dorney. 2016. “Shape Optimization of Supersonic Turbines Using Response Surface and Neural Network Methods.” In *39th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics. Accessed September 1. http://arc.aiaa.org/doi/abs /10.2514/6.2001-1065.

Papila, Nilay, Wei Shyy, Lisa Griffin, Frank Huber, Ken Tran, and Helen McConnaughey. 2000. “Preliminary Design Optimization For A Supersonic Turbine For Rocket Propulsion.” http://ntrs.nasa.gov/search.jsp?R=20000094037.

Pareto, Vilfredo. 1971. *Manual of Political Economy*. New York: A.M. Kelley.

Parmee, I. C. 2004. “A Review of the Development and Application of Cluster Oriented Genetic Algorithms.” In *IUTAM Symposium on Evolutionary Methods in Mechanics*, edited by Tadeusz Burczyński and Andrzej Osyczka, 331–40. Solid Mechanics and Its Applications 117. Springer Netherlands. doi:10.1007/1-4020-2267-0_31.

Parsons, M. G., David J. Singer, and John A. Sauter. 2016. “A Hybrid Agent Approach For Set-Based Conceptual Ship Design.” Accessed September 1. http://nsgl.gso.uri.edu/mit/mitw99002/mitw99002_part3d.pdf.

Parunak, H. Van Dyke, A. C. Ward, and J. A. Sauter. 1998. “A Systematic Market Approach to Distributed Constraint Problems.” In *International Conference on Multi Agent Systems, 1998. Proceedings,* 455–56. doi:10.1109/ICMAS.1998.699283.

Parunak, H. Van Dyke, A Ward, M. Fleischer, and J. Sauter. 2016. “The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research.” Accessed September 1. http://www.abcresearch.org/papers /RAPPID99JAAMAS.pdf.

Parunak, H. Van Dyke, Allen C. Ward, John A. Sauter, and others. 1999. “The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems.” *AI EDAM* 13 (3): 217–234.

Pirzada, U. M., and V. D. Pathak. 2012. "Newton Method for Solving the Multi-Variable Fuzzy Optimization Problem." *Journal of Optimization Theory and Applications* 156 (3): 867–81. doi:10.1007/s10957-012-0141-3.

Poss, Michael. 2014. "Robust Combinatorial Optimization with Variable Cost Uncertainty." *European Journal of Operational Research* 237 (3): 836–45. doi:10.1016/j.ejor.2014.02.060.

Prescott, B. P., and T. LeBaron. 2003. "Making Optimal Design Decisions for next Generation Dispensing Tools." In *Simulation Conference, 2003. Proceedings of the 2003 Winter,* 2:1388–93 vol.2. doi:10.1109/WSC.2003.1261580.

Pugh, Stuart. 1991. *Total Design: Integrated Methods for Successful Product Engineering*. Wokingham, England ; Reading, Mass: Addison-Wesley Pub.

Ramu, Palaniappan, Samy Missoum, and Raphael Haftka. 2016. "A Convex Hull Approach for the Reliability-Based Optimization of Transient Dynamic Problems." In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. American Institute of Aeronautics and Astronautics. Accessed September 1. http://arc.aiaa.org/doi/abs/10.2514/6.2004-4618.

Rao, S. S. 1987. "Description and Optimum Design of Fuzzy Mechanical Systems." *Journal of Mechanisms, Transmissions, and Automation in Design* 109 (1): 126–32. doi:10.1115/1.3258776.

Rao, S. S., and A. K. Dhingra. 1991. "Applications of Fuzzy Theories to Multi-Objective System Optimization." http://ntrs.nasa.gov/search.jsp?R=19910006699.

Rao, S. S., K. Sundararaju, B. G. Prakash, and C. Balakrishna. 1992. “Multiobjective Fuzzy Optimization Techniques for Engineering Design.” *Computers & Structures* 42 (1): 37–44. doi:10.1016/0045-7949(92)90534-7.

“Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency and Quality: Steven C. Wheelwright: 9781451676297: Amazon.com: Books.” 2016. Accessed September 1. https://www.amazon.com/Revolutionizing-Product-Development-Quantum-Efficiency/dp/1451676298.

Ross, James, Tulio Sulbaran, Andrew Strelzoff, and Nan Wang. 2017. “Conceptual Dynamic Collision Model For the Open Source Building Environment for Simulation and Training (OSBEST).” Accessed February 23. http://world-comp.org/p2011/CGV3566.pdf.

Ross, Phillip J. 1995. *Taguchi Techniques for Quality Engineering*. 2 edition. New York: McGraw-Hill Professional.

Ruusunen, Jukka, and Raimo P. Hamalainen. 1989. “Project Selection by an Integrated Decision Aid.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 101–21. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_7.

Saaty, Thomas L. 1978. “Exploring the Interface between Hierarchies, Multiple Objectives and Fuzzy Sets.” *Fuzzy Sets and Systems* 1 (1): 57–68. doi:10.1016/0165-0114(78)90032-5.

Saaty, Thomas L. 1989. “Group Decision Making and the AHP.” In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor

Edward A. Wasil, and Assistant Professor Patrick T. Harker, 59–67. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_4.

———. 2008. “Decision Making with the Analytic Hierarchy Process.” *International Journal of Services Sciences* 1 (1): 83–98.

Sasaki, Daisuke, Masashi Morikawa, Shigeru Obayashi, and Kazuhiro Nakahashi. 2001. “Aerodynamic Shape Optimization of Supersonic Wings by Adaptive Range Multiobjective Genetic Algorithms.” In *Evolutionary Multi-Criterion Optimization*, edited by Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne, 639–52. Lecture Notes in Computer Science 1993. Springer Berlin Heidelberg. doi:10.1007/3-540-44719-9_45.

Sasaki, Daisuke, Shigeru Obayashi, and Kazuhiro Nakahashi. 2002. “Navier-Stokes Optimization of Supersonic Wings with Four Objectives Using Evolutionary Algorithm.” *Journal of Aircraft* 39 (4): 621–29. doi:10.2514/2.2974.

Sen, Professor Pratyush, and Dr Jian-Bo Yang. 1998a. “An Integrated Multiple Criteria Decision Support System.” In *Multiple Criteria Decision Support in Engineering Design*, 211–41. Springer London. doi:10.1007/978-1-4471-3020-8_6.

———. 1998b. “MCDM and the Nature of Decision Making in Design.” In *Multiple Criteria Decision Support in Engineering Design*, 13–20. Springer London. doi:10.1007/978-1-4471-3020-8_2.

———. 1998c. “Multiple Attribute Decision Making.” In *Multiple Criteria Decision Support in Engineering Design*, 21–112. Springer London. doi:10.1007/978-1-4471-3020-8_3.

———. 1998d. “Multiple Criteria Decision Making and Genetic Algorithms.” In *Multiple Criteria Decision Support in Engineering Design*, 176–210. Springer London. doi:10.1007/978-1-4471-3020-8_5.

———. 1998e. “Multiple Objective Decision Making.” In *Multiple Criteria Decision Support in Engineering Design*, 113–75. Springer London. doi:10.1007/978-1-4471-3020-8_4.

———. 1998f. “Past, Present and the Future.” In *Multiple Criteria Decision Support in Engineering Design*, 242–55. Springer London. doi:10.1007/978-1-4471-3020-8_7.

Seppanen, Ville, Jukka Heikkila, and Katja Liimatainen. 2009. “Key Issues in EA-Implementation: Case Study of Two Finnish Government Agencies.” In *2009 IEEE Conference on Commerce and Enterprise Computing*, 114–120. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5210807.

Shafer, Glenn. 1976. *A Mathematical Theory of Evidence*. Princeton University Press.

Singer, D. J., and M. G. Parsons. 2003. “‘Evaluation of the Effectiveness of a Fuzzy Logic Software Agent to Aid Design Team Negotiation and Communication.” In *Proceedings of the 2nd International Conference on Computer Applications and Information Technology in the Maritime Industries, Hamburg, Germany*.

Singer, David J., Norbert Doerry, and Michael E. Buckley. 2009. “What Is Set-Based Design?” *Naval Engineers Journal* 121 (4): 31–43.

Sobek, D. K. Ii, A. C. Ward, and J. K. Liker. 1999. “Toyota’s Principles of Set-Based Concurrent Engineering.” *ResearchGate* 40 (2).

https://www.researchgate.net/publication/248139929_Toyota's_Principles_of_Set-Based_Concurrent_Engineering.

Soyster, A. L. 1973. "Technical Note—Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming." *Operations Research* 21 (5): 1154–57. doi:10.1287/opre.21.5.1154.

Steponavice, I., and K. Miettinen. 2016. "Survey On Multiobjective Robustness for Simulation-Based Optimization." Accessed September 1. http://users.monash.edu.au/~ingridas/ISMP2012_talk.pdf.

Taguchi, G., and M. S. Phadke. 1989. "Quality Engineering through Design Optimization." In *Quality Control, Robust Design, and the Taguchi Method*, edited by Khosrow Dehnad, 77–96. Springer US. doi:10.1007/978-1-4684-1472-1_5.

Technology, Copyright © Massachusetts Institute of, and 1977-2016 All rights reserved. 2016a. "The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster." *MIT Sloan Management Review*. Accessed September 1. http://sloanreview.mit.edu/article/the-second-toyota-paradox-how-delaying-decisions-can-make-better-cars-faster/.

———. 2016b. "Toyota's Principles of Set-Based Concurrent Engineering." *MIT Sloan Management Review*. Accessed September 1. http://sloanreview.mit.edu /article/toyotas-principles-of-setbased-concurrent-engineering/.

Thurston, Deborah L. n.d. "A Formal Method for Subjective Design Evaluation with Multiple Attributes." *Research in Engineering Design* 3 (2): 105–22. doi:10.1007/BF01581343.

Thurston, Deborah L., and Yun Qi Tian. 2013. "A Method for Integrating Utility Analysis into an Expert System for Design Evaluation." *arXiv Preprint arXiv:1303.5755*. http://arxiv.org/abs/1303.5755.

Tone, Kaoru, and Shigeru Yanagisawa. 1989. "Site Selection for a Large Scale Integrated Circuits Factory." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 242–50. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_16.

Tribus, Myron. 2016. *Rational Descriptions, Decisions and Designs: Pergamon Unified Engineering Series*. Elsevier.

Unal, Resit, and Edwin B. Dean. 1990. "Taguchi Approach to Design Optimization for Quality and Cost: An Overview." http://ntrs.nasa.gov/search.jsp?R=20040121019.

"USS Cole Bombing." 2016. *Wikipedia*. December 30. https://en.wikipedia.org/w/index.php?title=USS_Cole_bombing&oldid=757440999.

Vaidyanathan, Rajkumar, Tushar Goel, Raphael Haftka, Nestor Quiepo, Wei Shyy, and Kevin Tucker. 2004. "Global Sensitivity and Trade-Off Analyses for Multi-Objective Liquid Rocket Injector Design." In . American Institute of Aeronautics and Astronautics. doi:10.2514/6.2004-4007.

Vaidyanathan, Rajkumar, Nilay Papita, Wei Shyy, P. Kevin Tucker, Lisa W. Griffin, Raphael Haftka, Norman Fitz-Coy, and Helen McConnaughey. 2000. "Neural

Network and Response Surface Methodology for Rocket Engine Component Optimization." http://ntrs.nasa.gov/search.jsp?R=20000089909.

Vaidyanathan, Rajkumar, P. Kevin Tucker, Nilay Papila, and Wei Shyy. 2004. "Computational-Fluid-Dynamics-Based Design Optimization for Single-Element Rocket Injector." *Journal of Propulsion and Power* 20 (4): 705–17. doi:10.2514/1.11464.

Vargas, Luis G., and J. Bernat Roura-Agusti. 1989. "Business Strategy Formulation for a Financial Institution in a Developing Country." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 251–65. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_17.

Vlahakis, John G., and William R. Partridge. 1989. "Assessment of Security at Facilities That Produce Nuclear Weapons." In *The Analytic Hierarchy Process*, edited by Professor Bruce L. Golden, Assistant Professor Edward A. Wasil, and Assistant Professor Patrick T. Harker, 182–91. Springer Berlin Heidelberg. doi:10.1007/978-3-642-50244-6_12.

Wang, Jiachuan, and Janis Terpenny. n.d. "Interactive Evolutionary Solution Synthesis in Fuzzy Set-Based Preliminary Engineering Design." *Journal of Intelligent Manufacturing* 14 (2): 153–67. doi:10.1023/A:1022947329200.

Wang, Ten-See, and Yen-Sen Chen. 1990. "A Unified Navier-Stokes Flowfield and Performance Analysis of Liquid Rocket Engines." http://ntrs.nasa.gov/search.jsp?R=19900053583.

Ward, Allen C. 1989. "A Theory of Quantitative Inference for Artifact Sets Applied to a Mechanical Design Compiler." DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA216535.

Ward, Allen C., Jeffrey K. Liker, Durward K. Sobek, and John J. Cristiano. 1994. "Set-Based Concurrent Engineering and Toyota." In *Proceedings of ASME Design Engineering Technical Conferences, ASME*, 79–90.

Ward, Allen C., Tomas Lozano-Perez, and Warren P. Seering. 1990. "Extending the Constraint Propagation of Intervals." *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* 4 (1): 47–54.

Warner, Gary. 2005. "'PEO Ships Brief to NDIA." In *Expeditionary Warfare Division Annual Meeting, Fort Meyer, VA*. http://proceedings.ndia.org/5860/5860_Warner.pdf.

Wilson, Benjamin, David Cappelleri, Timothy W. Simpson, and Mary Frecker. 2001. "Efficient Pareto Frontier Exploration Using Surrogate Approximations." *Optimization and Engineering* 2 (1): 31–50.

Winkler, Robert L. 2003. *An Introduction to Bayesian Inference and Decision*. Probabilistic Pub.

Womack, James P., Daniel T. Jones, and Daniel Roos. 2007. *The Machine That Changed the World: The Story of Lean Production-- Toyota's Secret Weapon in the Global Car Wars That Is Now Revolutionizing World Industry*. Reprint edition. New York, NY: Free Press.

Wood, Kristin L., and Erik K. Antonsson. 1989. “Computations with Imprecise Parameters in Engineering Design: Background and Theory.” *Journal of Mechanisms, Transmissions, and Automation in Design* 111 (4): 616–625.

Xiao, Renbin, Zhengying Cai, and Xinhui Zhang. 2012. “A Production Optimization Model of Supply-Driven Chain with Quality Uncertainty.” *Journal of Systems Science and Systems Engineering* 21 (2): 144–60. doi:10.1007/s11518-011-5184-8.

Yager, Ronald, and David Basson. 1975. “Decision Making with Fuzzy Sets.” *Decision Sciences* 6 (3): 590–600. doi:10.1111/j.1540-5915.1975.tb01046.x.

Yager, Ronald R. 1978. “Fuzzy Decision Making Including Unequal Objectives.” *Fuzzy Sets and Systems* 1 (2): 87–95. doi:10.1016/0165-0114(78)90010-6.

Young, Rosalind Cecily. 1931. “The Algebra of Many-Valued Quantities.” *Mathematische Annalen* 104 (1): 260–90. doi:10.1007/BF01457934.

Yu, H., and H. M. Liu. 2012. “Robust Multiple Objective Game Theory.” *Journal of Optimization Theory and Applications* 159 (1): 272–80. doi:10.1007/s10957-012-0234-z.

Zadeh, L. A. 2016. “Fuzzy Sets-Information and Control-1965.pdf.” Accessed September 1. https://people.eecs.berkeley.edu/~zadeh/papers/Fuzzy%20Sets-Information%20and%20Control-1965.pdf.

Zhou, Guan, Zheng-Dong Ma, Guangyao Li, Aiguo Cheng, Libin Duan, and Wanzhong Zhao. 2016. “Design Optimization of a Novel NPR Crash Box Based on Multi-Objective Genetic Algorithm.” *Structural and Multidisciplinary Optimization* 54 (3): 673–84. doi:10.1007/s00158-016-1452-z.

Zimmermann, H. J., and H. J. Sebastian. 1993. "Optimization and Fuzziness in Problems of Design and Configuration." In , *Second IEEE International Conference on Fuzzy Systems, 1993*, 1237–40 vol.2. doi:10.1109/FUZZY.1993.327569.

———. 1994. "Fuzzy Design-Integration of Fuzzy Theory with Knowledge-Based System-Design." In , *Proceedings of the Third IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence*, 352–57 vol.1. doi:10.1109/FUZZY.1994.343673.

Zimmermann, H.-J. 2001. *Fuzzy Set Theory—and Its Applications*. Dordrecht: Springer Netherlands. http://link.springer.com/10.1007/978-94-010-0646-0.