The Thesis Committee for Jordan Lauren Matthews
Certifies that this is the approved version of the following thesis:

A Bayesian Network Classifier for Quantifying Design and Performance Flexibility with Application to a Hierarchical Metamaterial Design Problem

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Carolyn C. Seepersad

Michael R. Haberman

A Bayesian Network Classifier for Quantifying Design and Performance Flexibility with Application to a Hierarchical Metamaterial Design Problem

by

Jordan Lauren Matthews, B.S.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2013

Dedication

To Science!

Acknowledgements

I would first and foremost like to acknowledge and thank my advisor, Dr. Carolyn Seepersad, for her guidance and support since I came to the University of Texas. Dr. Seepersad was an ideal role model for me – she has helped shape my work ethic as well develop a healthy work/life balance. Whether it was spending many long hours reviewing my research progress in order to give me exhaustive feedback, or inviting me for a burger to relieve some stress, Dr. Seepersad gave me no excuses to achieve anything short of excellence. I would also like to thank Dr. Haberman's help in developing my thesis, and especially thank him for his willingness to teach me about subjects inside, and outside the scope of this research.

I am very thankful of everyone that I was able to interact with over the past several years at the University of Texas. I would like to thank my fellow lab mates for their guidance in my transition to becoming a successful graduate student. I would like to particularly acknowledge the hard work done my fellow lab mate, Tim Klatt, for his help in developing and sharing the code that I used for the metameterial design problem in my thesis. I am also very thankful for the opportunity to interact with and learn from Dr. David Shahan. The research in my thesis largely extends the prior work done by Dr. Shahan for his PhD, and without his help this work would not have been possible.

Finally, I would like to thank my family – my mom for fostering my interest in science since before I can remember, and my dad for helping me maintain a proper outlook for having a happy and fruitful life.

Abstract

A Bayesian Network Classifier for Quantifying Design and Performance Flexibility with Application to a Hierarchical Metamaterial Design Problem

Jordan Lauren Matthews, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Carolyn C. Seepersad

Design problems in engineering are typically complex, and are therefore decomposed into a hierarchy of smaller, simpler design problems by the design management. It is often the case in a hierarchical design problem that an upstream design team's achievable performance space becomes the design space for a downstream design team. A Bayesian network classifier is proposed in this research to map and classify a design team's attainable performance space. The classifier will allow for enhanced collaboration between design teams, letting an upstream design team efficiently identify and share their attainable performance space with a downstream design team. The goal is

that design teams can work concurrently, rather than sequentially, thereby reducing lead time and design costs.

In converging to a design solution, intelligently narrowing the design space allows for resources to be focused in the most beneficial regions. However, the process of narrowing the design space is non-trivial, as each design team must make performance trade-offs that may unknowingly affect other design teams. The performance space mapping provided by the Bayesian network classifier allows designers to better understand the consequences of narrowing the design space. This knowledge allows design decisions to be made at the system-level, and be propagated down to the subsystem-level, leading to higher quality designs.

The proposed methods of mapping the performance space are then applied to a hierarchical, multi-level metamaterial design problem. The design problem explores the possibility of designing and fabricating composite materials that have desirable macro-scale mechanical properties as a result of embedded micro-scale inclusions. The designed metamaterial is found to have stiffness and loss properties that surpass those of conventional composite materials.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Effective design of complex systems requires proper organization, starting from the first day of the design process. The complex design problem must first be decomposed into a hierarchy of smaller, more manageable design problems. A design team is assigned responsibility for each decomposed design problem, and must be coordinated by top-level management to work concurrently or sequentially with each other. Each design team must meet performance criteria unique to each team, while collectively meeting top-level performance criteria. The process of then converging to a final satisfactory design is non-trivial, as each design team must make performance trade-offs that may unknowingly affect other design teams. In this thesis, a tool is developed to allow design teams to accurately map and communicate knowledge of their design spaces and performance capabilities as a means of collaborating with other designers.

1.1 The Modern Engineering Designer's Perspective

Everyday engineering companies strive to deliver higher quality products at low costs to the consumer, in order to stay competitive. Consumers now demand that a product meet a wide range of customer performance criteria, leading to crucial decisions of designing "one-size-fits-all" products or allowing consumers to purchase customized products. At the same time, engineers are forced to meet compressed timelines in order to shorten the product's design time. These challenges must be overcome while mitigating the never-ending uncertainty that is inherent in every stage of the design process.

Although the human race has repeatedly shown the intellectual capability of designing extremely complex and beneficial products such as airplanes and skyscrapers, the frequency of product failures brings the age-old advertiser's expression to mind: "We design our products fast, good, and cheap. You may choose any two qualities."

Many of the difficulties in engineering design are the result of a constant increase in product complexity. An increase in product complexity has especially been seen over the past 100-200 years due to vast improvements in manufacturing and communication. Inventor Ray Kursweil famously recognized that technological evolution follows an exponential growth because the latest technology is used to create the next generation products (Kursweil, 2008). Advances in engineering numerical simulation software, such as Finite Element Analysis, which may currently require hours or days to complete a single simulation, were unimaginable a half a century ago. Correspondingly, a simulation that took days to run a decade ago can now be completed in minutes. Engineers now have the capability to use tools that were once a "final check" before prototyping, as tools to find better designs.

No longer is it the case that a single engineering team can design a large-scale system on their own, as a wide range of varying expertise is necessary. While there is not a universal definition of product complexity, several characteristics are common to complex products. In addition to difficulty and novelty, complexity in engineering systems design is characterized as having high interdependencies between disciplines/departments, as well as having a high number of parts, functions, and disciplines (Kim & Wilecon, 2009). The quintessential example of a large-scale complex

system is the modern-day airplane, which has recently seen dramatic complexity increases. The F-16 fighter jet, which was developed in 1974, had 15 subsystems and an order of 10^3 interfaces, while the F-35 fighter jet, which completed development in 2006, had 130 subsystems and an order of 10^5 interfaces (Becz *et al*., 2010). As a result of this complexity increase, the development of Lockheed Martin's F-35 has had production delays of 1-3 years and an unexpected increase of 7%-10% in its lowest rate unit cost (Gertler, 2012). The failings in the F-35 design process have been shared by commercial aircraft design processes as well. Airbus's A380 overran its scheduled design time by 18 months, which was still better than Boeing's development of the 787 Dreamliner, which saw production delayed by 28 months (Becz *et al*., 2010).

These recent design process inadequacies represent a multitude of interlaced flaws, many of which are unique to each design process. However, acting Pentagon Acquisition Chief, Frank Kendall, specifically recognized Lockheed Martin's inability to effectively work concurrently between design teams as the fundamental cause for delays (Sweetman, 2012). Kendall added that Lockheed Martin held far too optimistic expectations that satisfactory performance criteria would be met after compiling the design selections from each design team. Nevertheless, if indeed Lockheed Martin held overly optimistic expectations of their system-level performance capabilities, the question remains: "What performance values *should* have been expected?"

The designer's tool presented in this thesis will give design teams at every hierarchical level the means to accurately communicate their performance capabilities with each other. High-level design teams can utilize this knowledge to make performance

tradeoff decisions for design teams lower on the decomposition hierarchy in order to best benefit the overall performance of the product. Additionally, in an effort to efficiently converge on a final, overall design, a method to quantify each design team's flexibility is drawn from this tool.

1.2 Enhancing Collaboration in Set-Based Design

As design engineers work to converge to a single solution, adopting a set-based strategy in which design teams maintain flexibility by sharing ranged sets of solutions (as opposed to point solutions), is a widely accepted strategy to mitigate the effects of unforeseen events in the design process. A set-based design process allows design teams to delay decision making until their design space is better understood, giving collaborating design teams more accurate depictions of each other's knowledge at the current state. Game theoretic techniques have been applied to show that design processes in which design teams poorly communicate their design objectives will converge to a sub-optimal solution point, called the "Nash Equilibrium" (Gurnani & Lewis, 2008). Thus, creating a design process environment wherein design teams can collaborate on design decision making can lead to improved system-wide performances.

In order to better understand the benefits of concurrent engineering, a brief discussion on terminology is presented here, in preparation for the remainder of this thesis. The variables controlled by design teams, such as the diameter of a rod, are termed *design variables*. Similarly, the specific features for gauging the quality of a design, such as motor horsepower or efficiency, are termed *performance variables*. The set of all

possible design variable combinations is termed the *design space*, and is typically made finite by assigning lower and upper bounds to each design variable. The set of all possible performance variable combinations is termed the *performance space*. The finite subset of the performance space that can be mapped from the bounded design space is termed the *attainable* performance space. Within the attainable performance space designers seek to find designs that meet certain criteria or thresholds; for example, management may state that the fuel efficiency must be greater than 38 miles per gallon. The set of performance variables that meet all performance criteria is termed the *satisfactory* performance space, and the exploitation of this subspace is the fundamental task of all design processes. The challenge in design is to develop an understanding of the relationships between design variables, specifically, the relationships that correspond to designs within the satisfactory performance space.

The recent concurrent design process failures in industry have stemmed from poor communication between design teams, and have resulted in highly iterative point-based design processes. The point-based design process approach can be visualized as sequential decision making, in which design teams optimize their controlled design variables and only share their latest design configuration with each other. In contrast, a concurrent set-based design process approach can be thought of as design teams working in parallel, sharing information in order to minimize the chance of design iterations. These approaches are illustrated in Figures 1.1 and 1.2.

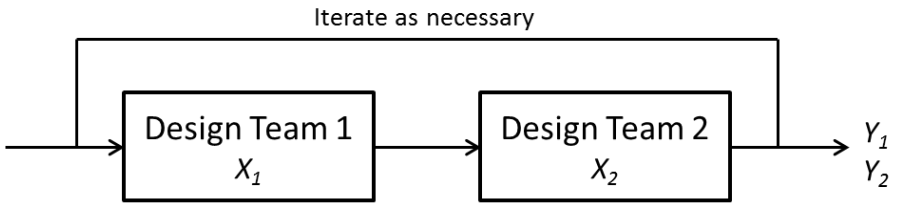

Figure 1.1. A depiction of a point-based design process approach where design teams work in sequence.

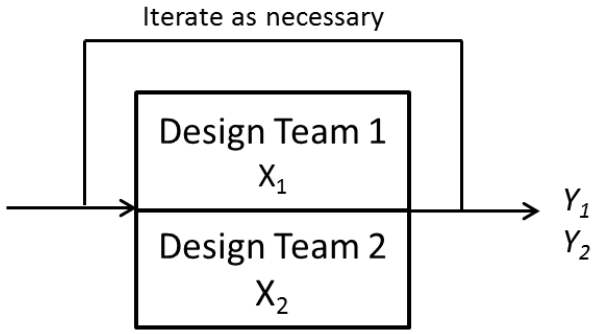

Figure 1.2. A depiction of a concurrent set-based design process approach where design teams work in parallel.

The results of a hypothetical design process that adopts a point-based approach are illustrated in Figure 1.3. In this example, there are two design teams, each controlling one design variable and one performance variable. The goal of the design teams is to iterate until a solution is found that meets both design teams' criteria for satisfactory performance. The first design found that meets the performance acceptability criteria can conclude the design process; however, a better design can likely be found if resources permit further iteration. The point-based design process results in 5 design iterations before a mutually satisfactory design was found, and 7 iterations before a superior design

was agreed upon by both parties. Although there are several methods for optimizing multilevel, decomposed design problems, such as Analytical Target Cascading and Concurrent Sub Space Optimization, they can be computationally expensive and require precise system-wide coordination and management that is often unattainable in non-automated processes, which quickly devolve into a guess-and-check method (H. M. Kim, Michelena, Papalambros, & Jiang, 2003; Wujek, Renaud, Batill, & Brockman, 1996). The key takeaway of a point-based design approach is that it necessitates design iterations, which can become the primary cause of developmental delays in later stages of the design process.

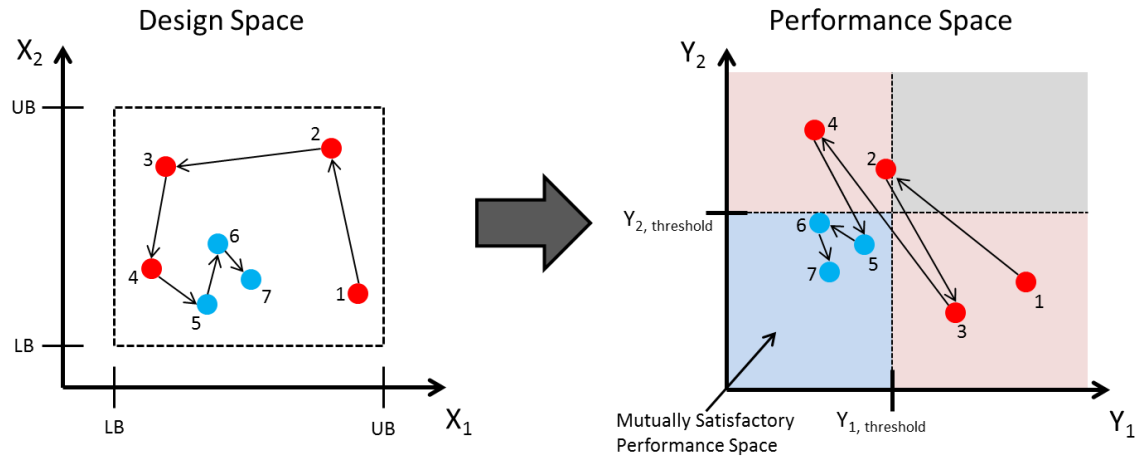

Figure 1.3. The design space (left) and performance space (right) of a point-based design process.

Set-based collaborative design is based on spending extra time and resources early in the design process, in order to reduce the chance of iterations in later phases of the design process. The benefits of an effective collaborative design process are

exemplified by Toyota's adoption of set-based concurrent engineering practices in the 1990's, which led to remarkable reductions in lead time and design costs (Clark & Fujimoto, 1991; Sobek, Ward, & Liker, 1999; Ward, Liker, Cristiano, & Sobek, 1995). A research study by Sobek et al. identifies three key principles of Toyota's concurrent engineering practices as the basis of their success: (1) Map the design space, (2) Integrate by intersection, and (3) Establish feasibility before commitment (Sobek *et al*., 1999). Essentially, these principles address the most basic goal of developing the best product that can be feasibly created. However, the basic nature of these principles emphasizes the difficulty of a large-scale concurrent design process.

The capability of a set-based design approach to excel at the principles proposed by Sobek *et al*. can be understood by comparing a hypothetical set-based concurrent design process with the point-based design process shown in Figure 1.3. The same example design process is shown again in Figure 1.4, but with the design teams sharing sets of information, as opposed to single design points. In the examples shown in Figures 1.3 and 1.4, the two performance variables, y_1 and y_2, are attempted to be minimized, with the target values $y_{1,threshold}$ and $y_{2,threshold}$, representing the maximum satisfactory performance value for design team 1 and 2, respectively. Following the principles proposed by Sobek *et al*., each design team first samples its own design space to understand performance tradeoffs and their locations in the design space. As each design team continues to sample design alternatives, sharing the sets of designs values that are known to produce feasible and satisfactory results focuses the search on designs that are known to have superior performance characteristics. Through the process of sampling

design alternatives, a mapping of the performance space is created, and the attainable performance space can be distinguished. Once an accurate mapping of the attainable performance space has been created, the two design teams simply need to look within the intersection of each other's satisfactory design spaces to search for a final solution.

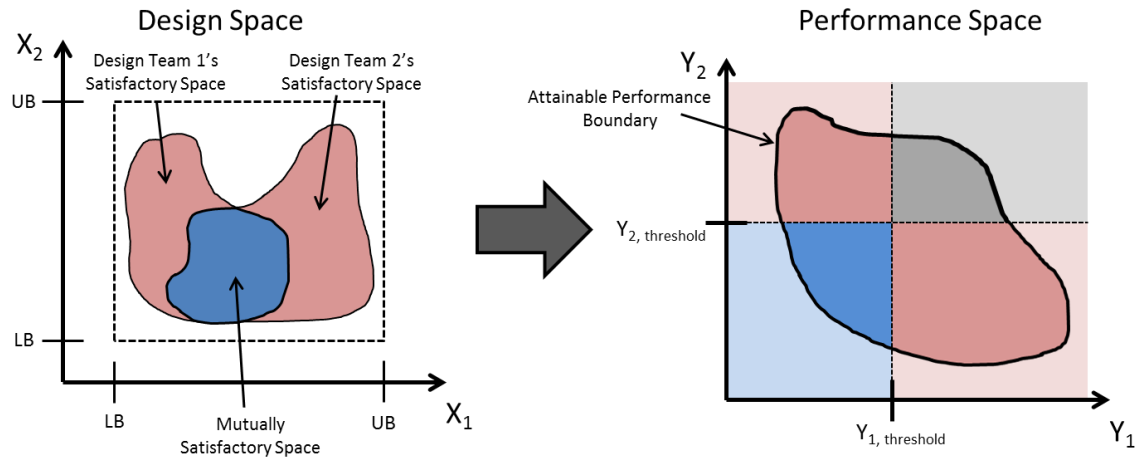

Figure 1.4. The design space (left) and performance space (right) of a set-based design process.

By helping designers better understand where promising regions of the design space exist before making constricting design decisions, set-based design also allows for the mitigation of unexpected occurrences in the design process. Unexpected occurrences are common in large-scale design processes, and may be the result of changing performance requirements, imprecision in design decisions, and even mistakes made by fellow designers (Antonsson & Otto, 1995; Devendorf & Lewis, 2008; Ilkka, 1985). The fundamental reason that a set-based design approach can adapt to changes in both design

space constraints as well as performance criteria, is that *flexibility* is maintained until the late stages of the design process.

Flexibility has many connotations in design research, but is generally regarded as "a property that promotes change in both the design and performance spaces" (Ferguson, Siddiqi, Lewis, & de Weck, 2007). Additionally, flexibility has separate meanings in the design and performance spaces. In the design space, *design flexibility* is defined for this research as the size of the subspace that produces satisfactory designs (Suh, 1990). Essentially, design flexibility can be understood as the degree to which a design can be changed while still remaining in the satisfactory performance space (Simpson, Rosen, Allen, & Mistree, 1996). Similarly in regards to the performance space, *performance flexibility* is defined as the size of the satisfactory performance space associated with a specific set of designs. Performance flexibility can be thought of as a measure of insensitivity to changes in performance criteria. Sets of designs with high performance flexibility can meet a wide range of performance requirements, and thus easily adapt to changes in performance criteria.

Taguchi's robust design method, published in 1993, is a very commonly used method to develop high quality products by leveraging flexibility (Chang, Ward, Lee, & Jacox, 1994; Taguchi & Cariapa, 1993). However, in complex design problems it is difficult to determine how much flexibility is needed. Furthermore, it is even more difficult to quantitatively determine if a design has an appropriate amount of flexibility. A method to quantify flexibility has been recognized as an insufficiently answered research question for over two decades, and a flexible systems review paper stated in 2007 that a

standardized approach to quantifying flexibility still needs to be discovered *(Ferguson et al*., 2007; Gupta & Goyal, 1990; Toni & Tonchia, 1998). Several approaches to quantifying design and performance flexibility are reviewed in chapter two, but it should be noted that all of the previously proposed approaches have significant deficiencies in practice.

A fundamental difficulty in approximation of the satisfactory design and performance spaces is that they must be done based on limited information. Obviously taking a copious number of sample points can make the size and shape of this space clear; however, engineers rarely have the computing power or time to perform an exhaustive sampling process. A hypothetical performance space is shown in Figure 1.5, and illustrates the challenges in developing a standardized method for mapping and quantifying regions that are known to be arbitrarily shaped. The most common approach to approximating the performance space is to use intervals (Chen & Ward, 1995). However, complex relationships that exist in engineering can rarely be captured by rectangular shape. Figure 1.5 depicts two intervals, one being overly generous in approximating the design's performance capabilities, and the other being overly conservative. The overly generous interval is seen to misclassify portions of the unattainable performance region as feasible, and conversely, the overly conservative is seen to misclassify portions of the feasible performance space as either unattainable or infeasible. The obvious augmentation to interval classification, to improve its accuracy, is to divide the performance space into many separate intervals or bins. However, classification using bins is also inherently limited because it is a very simplistic

representation of the design space that does not capture confidence levels of the classification or guide subsequent sampling of the design space.

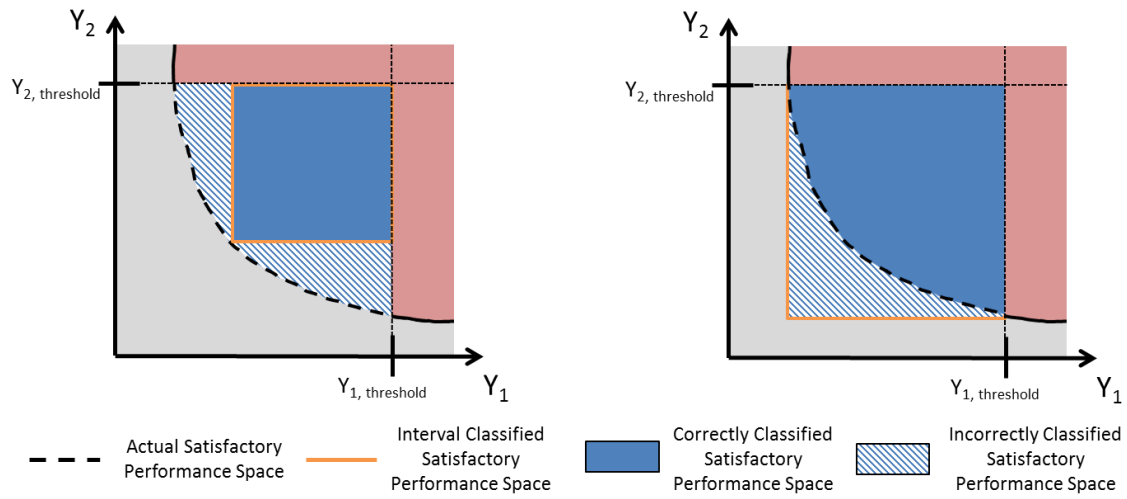

Figure 1.5. Two performance spaces showing the error resulting from a conservative (left), and generous (right) interval approximation.

A recently proposed method for classifying arbitrarily shaped regions of the design space uses a Bayesian network classifier to capture the knowledge of the design space gained from sample points (Shahan & Seepersad, 2012). To approximate the decision boundary between the satisfactory and unsatisfactory regions in the design space Shahan and Seepersad create a probability density function for each class (satisfactory and unsatisfactory). The probability density function is created by assigning a kernel of normal distribution on each sample point, and the kernels are subsequently summed over all sample points for each class. An unknown design point is classified as satisfactory if the class conditional probability density function for the satisfactory class of sample

points is greater than that of the unsatisfactory class at that point. The method can classify regions of arbitrary shape and connectivity, and it can guide sequential sampling and search. A more in depth discussion of the Bayesian network classifier method, as well as other methods for classification is given in Chapter 2.

The research presented in this thesis extends the work of Shahan and Seepersad, and uses a Bayesian network classifier to map arbitrarily shaped regions of the *performance space*. The major difference between the design and performance spaces, which is addressed by the method proposed in this thesis, is that while all points in the design space can be linked to points in the performance space, the converse is rarely true (Klein, Sayama, Faratin, & Bar-Yam, 2003). Therefore, in addition to differentiating the feasible and infeasible regions of the performance space, it is also necessary to distinguish the attainable versus unattainable regions of the performance space. These extensions of Shahan and Seepersad's work are required for multilevel design, in which top-down, performance-driven, multilevel design requires back propagation across multiple levels.

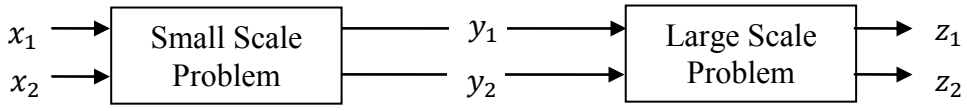

Figure 1.6. An example of a multi-level design problem.

In the multilevel design problem shown in Figure 1.6, the vector of design variables, $\boldsymbol{x}$, is the input to a small scale problem, whose output, $\boldsymbol{y}$, is the input to a large scale problem, with performance parameters, $\boldsymbol{z}$. This design process structure is common

in engineering and is interesting because the design space of the large scale design team is defined by the feasible performance space of the small scale design team. Figure 1.7 shows for a two dimensional problem, how large scale candidate designs, ***y***, are classified according to their large scale performance, ***z***, and that classification is then used to identify high-performance candidate designs, ***x***, at the small scale.

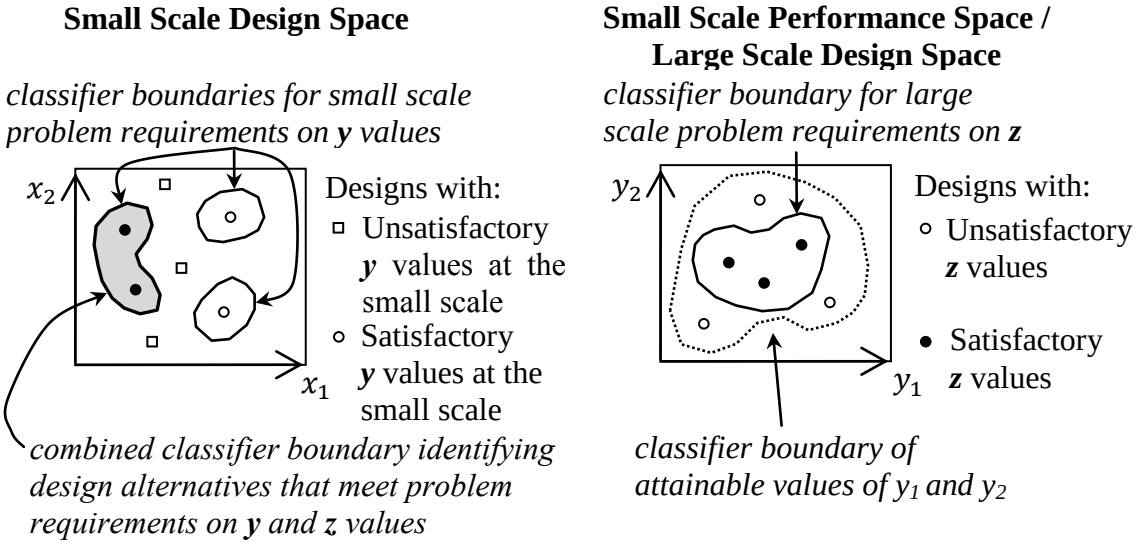

Figure 1.7. The backward propagation of requirements across levels of a multi-level design problem.

Complex systems are inherently multilevel design problems, having system-level performance requirements, as well as subsystem performance targets that must first be met. For the subsystem design team, it is initially sufficient to map the satisfactory design space, comprised of combinations of *independent* design variable values that offer satisfactory performance with respect to one or more dependent performance parameters. Subsequently in order to solve the system-level design problem, it is also necessary to

map the subsystem's satisfactory performance space of *dependent* variable values, as those dependent variables also serve as input to system-level design. For example, in Figures 1.6 and 1.7, the output of the small scale problem, $\boldsymbol{y}$, also serves as input to the large scale problem. Accordingly, it is important to create a design space mapping of the combinations of y_1 and y_2 values that provide satisfactory values of $\boldsymbol{z}$ at the large scale, but also a performance space mapping of the combinations of y_1 and y_2 values that can be feasibly achieved by varying x_1 and x_2 at the small scale, so that the design and performance space mappings of y_1 and y_2 values can be intersected during the design exploration process.

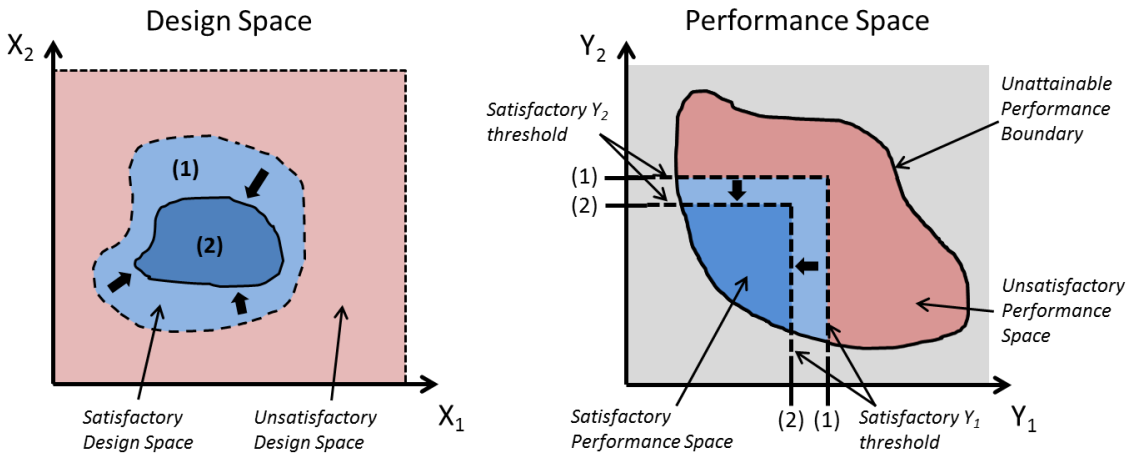

Figure 1.8. An example of before (1) and after (2) narrowing the design space by constricting the satisfactory performance threshold values.

To conclude a design process, the design space must be narrowed to a single, satisfactory point before a product can be made. As it is impossible to have complete knowledge of the overall design space, narrowing the design space allows a design team

to concentrate its resources to develop a better understanding of that smaller region (Wood & Agogino, 2005). Narrowing the design space is ideally equivalent to tightening the performance targets, as shown in Figure 1.8, leading to a narrowed satisfactory performance space. Timing is critical when narrowing the design space, as narrowing it too quickly can possibly exclude the most desirable regions of the design space, especially if the interactions between design variables are not well understood yet. Furthermore, in a multilevel design problem, the design decisions made by lower level design teams may make it more difficult, or impossible for an upper level design team to find a satisfactory design. In the case that a downstream design team cannot find a satisfactory design, iterations must take place between the design teams, increasing the product lead time (Shahan & Seepersad, 2010). An accurate mapping of the performance space allows a design team to intelligently narrow its design space, by understanding exactly how its design decisions restrict its *design* and *performance flexibility*.

1.3 Research Overview

The goal of this research is to create a tool that designers can use to better understand the performance capabilities of a design, based on the results of experiments and/or computational simulations. Using a Bayesian network classifier, a method is proposed to map the desirable regions of the performance space. The performance space mapping is then used to quantify performance flexibility. The research objectives for this thesis are presented in Table 1.1.

Table 1.1. Research Objectives	
1.	Provide designers with a tool to map and classify arbitrarily shaped regions of the performance space
2.	Provide designers with a tool to autonomously quantify the flexibility of the design and performance space
3.	Demonstrate the utility of design and performance space Bayesian network classifiers, in a multi-level design case study

The methods presented in this thesis form a tool to allow designers to better address Sobek *et al*.'s three fundamental set-based design principles: (1) Map the design space, (2) Integrate by intersection, and (3) Establish feasibility before commitment (Sobek *et al*., 1999). The first and second guidelines are addressed by the first research objective in Table 1.1: the creation of a method to map the performance space, as one design team's performance space can often represent a downstream design team's design space. Additionally, as seen in Figure 7, the mapping tool allows design teams to intersect one another's design spaces in order to find system level satisfactory designs. The third design principle is addressed by the second research objective in Table 1.1: the creation of a tool to quantify performance flexibility, which can be used to intelligently determine when to fix design variables once the performance capabilities are well understood.

The research objectives are explored in a series of chapters in this thesis. The second chapter reviews the recent advances in concurrent engineering design research in order to understand how the research tools presented may impact different aspects of concurrent design. Specifically, advances in representing sets of designs and parameters are discussed, in addition to reviewing the most state-of-the-art methods to quantify flexibility. The third chapter addresses the first research objective in Table 1.1, by providing the necessary framework to map the performance space using Bayesian network classifiers. The fourth chapter addresses the second research objective in Table 1.1, by presenting several methods to quantify the size of the feasible regions of the design and performance space. The goal of the methods presented in Chapter 4 is to create robust convergence criteria, such that the design and performance flexibility quantification can be autonomously performed.

The fifth chapter applies the methods presented in Chapters 3 and 4 to a hierarchical design problem to examine its effectiveness. The design problem explores the possibility of designing and fabricating composite materials that have macroscopic mechanical stiffness and loss properties that surpass those of conventional composites. It has been previously theorized that this can be done by embedding small volume fractions of high loss micro-scale inclusions in a continuous host material (Lakes, 2001). Achieving high stiffness and loss from these materials by design, however, is a nontrivial task. To solve this design problem, a hierarchical multilevel material model is presented, coupled with a hierarchical design approach using Bayesian network classifiers to map the design and performance spaces at each hierarchical level. Length scales range from

the behavior of the structured microscale inclusions to the effective properties of mesoscale composite materials to the performance of an illustrative macroscale component, a vibrating cantilever beam that has been coated with the designed composite material.

Chapter 2: Foundational Research in Set-Based Design

Design teams that engage in a set-based design paradigm have been shown to produce higher quality products in shorter lead times, especially in the design of complex systems (Sobek, Ward, & Liker, 1999). A set-based design approach allows for better collaboration between design teams, and gives designers the opportunity to delay important design decisions until the impact of those decisions are better understood. A primary challenge in set-based design is defining the set of designs, as information about how design variable values correlate to performance values is typically incomplete and abstract (Pacheco, Amon, & Finger, 2003; Wood & Agogino, 2005). The focus of this thesis is to develop a means (1) to represent or map a set of designs accurately in both design and performance space and (2) to quantify the design and performance flexibility embodied in the set of designs. This chapter is segmented by these goals, and reviews several prominent techniques and methodologies previously presented to achieve these goals.

Section 2.1 reviews methods used to represent sets of designs in performance space. The method of using Bayesian network classifiers to map the design space is also described in this section, to highlight the necessary extensions required to use this method in representing sets of designs in performance space, which is the novel method presented in Chapter 3. Section 2.2 reviews methods to quantify the design and performance flexibility embodied in a set of designs.

2.1 Representing Sets of Performances

Methods to represent performance sets consider how design teams can use knowledge of the design space, obtained by experimentation, simulation, and expert knowledge, to represent or map the space of attainable performance. An accurate map of an attainable performance space has many benefits to a design process. One design team's set of attainable performances values could also be the set of designs that a downstream design team is able to choose from in a multilevel design problem. An inaccurate representation of an upstream design team's performance space can result in the downstream design team selecting a design that may not be attainable. Another use of mapping the set of attainable performance values is in the development of product families. A product family is akin to a set of performance capabilities that can be mapped from a set of platform-based products that share many of the same design variable values (Simpson, Maier, & Mistree, 2001).

Sets of performance parameter values can be represented by discrete values, continuous ranges, or discontinuous ranges, as well as with non-exact, probabilistic parameters, as shown in Figure 2.1. Discrete values are not effective at performance set representation because they do not give any insight on the attainability of in between values. Unfortunately, discrete attainable performance values are typically how engineers receive their information. Conducting a computationally intensive design analysis requires specifying a single design to be inputted, and results in a single performance output. For an effective capture of knowledge gained from simulations, the discrete sample values of the performance space must be extended to continuous or discontinuous

ranges. Uncertainty in the extension from discrete points to continuous or discontinuous ranges is the fundamental problem to be addressed in all set-based design methods. An overly generous extension between discrete performance sample points may lead designers to believe they can achieve performance values beyond their capable limits, and conversely an overly conservative extension between points may lead designers to abandon research in a technology for a lack of potential.

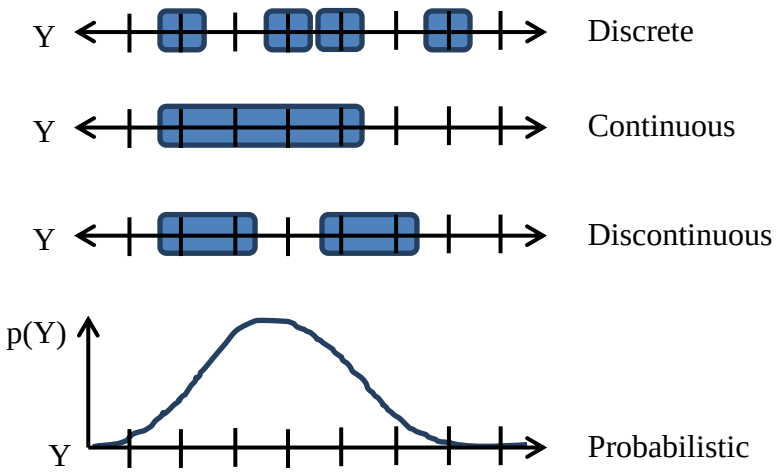

Figure 2.1. Several examples of set representations

2.1.1 Fuzzy Sets

One proposed method to mitigate the uncertainty associated with mapping the performance space is to allow the designers themselves to capture this uncertainty using fuzzy logic. The Method of Imprecision (MoI) uses fuzzy logic to capture a designer's preferences among performance values (Antonsson & Otto, 1995; Wood & Antonsson,

1989). This method attempts to improve trade-off decision making between design teams by allowing each design team to define a fuzzy preference function. The fuzzy preference function maps the performance space by assigning every performance point a value between zero and one, in which a value of zero corresponds to a certain unsatisfactory performance value, and a value of one corresponds to a certain satisfactory performance value. Attainability of the performance space is incorporated into the fuzzy preference function by assigning a value of zero to unattainable performance points. Design teams then can share performance variable sets by sharing the performance values that have non-zero preference function values.

A fuzzy preference function for the performance space allows design teams to intersect one another's preference functions to find a design with the highest likelihood of mutual satisfaction, thus minimizing the risk of additional iterations between design teams. Using fuzzy logic to define performance preferences has been combined with genetic algorithms, and has been shown capable of solving highly complex design problems with many design teams (Saridakis & Dentsoras, 2006). Mapping the performance space using fuzzy logic, however, does not aid in the determination of performance variable attainability, and therefore is typically combined with other set representation methods. In fact, the study done by Saradakis and Dentsoras assumes that each design team has complete knowledge of its attainable performance space, thus side-stepping the challenge of defining this boundary.

2.1.2 Interval Sets

The most common method to represent the performance space is by using interval sets – a method that gained notoriety during Toyota's set-based design studies (Ward, Liker, Cristiano, & Sobek, 1995). Intervals treat each variable independently, and by defining an upper and lower bound a continuous ranged set is created for each variable, as shown in Figure 2.2. Intervals do not provide an accurate representation of an irregularly shaped space, such as the one shown in Figure 2.1, and results in either an overly generous or conservative representation. Combining the intervals from each performance variable creates a mapping of the performance space, and forms a hyperrectangular ranged set (Chen & Ward, 1995). Intervals are very applicable to large scale, complex design problems by making use of interval calculus methods (Davis, 1987; Reddy & Mistree, 1992). Ward et al. extended these interval calculus methods into Labeled Interval Calculus (Ward, Lozano-Perez, & Seering, 1990), which served as a set of rules to cascade design and performance sets among design teams (Finch & Ward, 1997) and across hierarchical levels (Panchal & Allen, 2005).

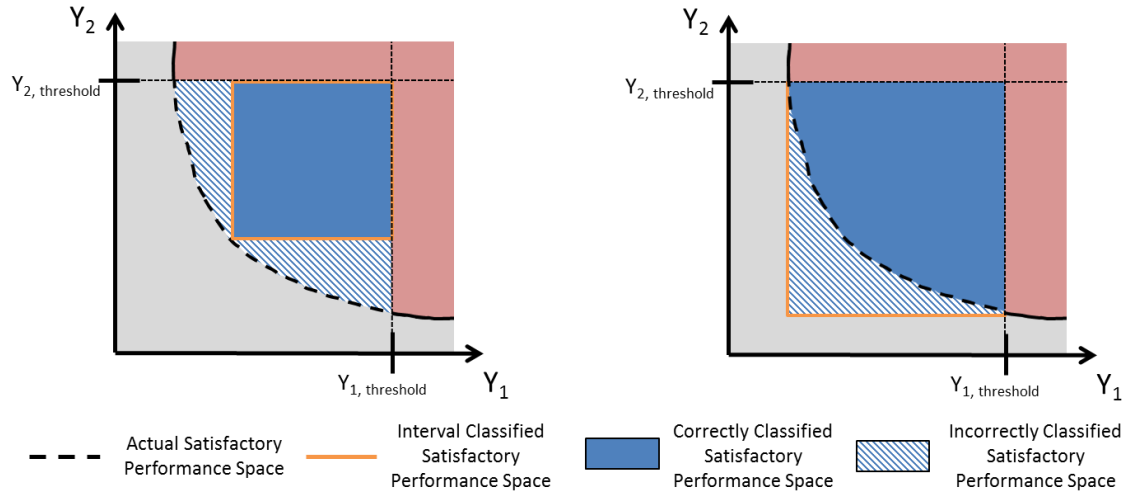

Figure 2.2. Two performance spaces showing the error resulting from a conservative (left), and generous (right) interval approximation.

Intervals can be created with various goals in mind, for example to minimize the number of falsely classified attainable performances in the interval, as shown in the left interval of Figure 2.2, or to minimize the number of falsely classified unattainable performances in an interval, as shown in the right interval of Figure 2.2. Interval techniques are subject to error, however, because intervals assume linear relationships between design variables, and have been thus shown to poorly capture feasible regions in the design space (Shahan & Seepersad, 2010). The Interval-based Constraint Satisfaction method (IBCS) proposed by Panchal *et al*. increases the effectiveness of interval classification, by beginning at a generous interval and slowly converging to a narrow interval as information about the system is gained (Panchal & Allen, 2005; Panchal, Gero Fernandez, Paredis, Allen, & Mistree, 2007). IBCS makes use of the set-based design

guideline of utilizing flexibility in the early stages of design, and the method presented in this thesis is targeted to be used in a similar manner, but with improved classification accuracy relative to intervals.

The ability of intervals to more accurately capture regions of attainability can be enhanced by decomposing each variable into interval bins. However, discretizing the performance space leads to assumptions on how one sample point's attainability can be extended to regions around it. The accuracy of interval bins is thus highly dependent on the size of the bins, as well as the shape of the actual attainable performance space. Liu e*t al*. proposed a method that incorporated clustering of performance points to create multiple, independent intervals (Liu, Chen, Scott, & Qureshi, 2008). The method proposed by Liu *et al*. significantly improves the accuracy of intervals, while still utilizing the benefits of interval calculus. Although this method can theoretically provide an accurate mapping at the limit of infinitesimally small interval bins, an inability to extrapolate the attainability of one training point to the region around it limits the minimum bin size in practice. Additionally, the assumption of variable independence induces an error that cannot be reduced by decreasing the bin sizes.

2.1.3 Probabilistic Sets

The uncertainty associated with mapping design and performance spaces is well suited for probabilistic methods. A design team can map a probability density functions (PDF) onto the performance space, to represent the likely attainability of performance variable values. Representing sets of design and performance values using PDF's allows

designers to assign a varying degree of belief that a particular design or performance point belongs in that set. PDF's can easily capture designer's uncertainty that may originate from manufacturing tolerances. Robust design techniques can be used to minimize uncertainty due to noise factors, and/or uncertainty due to variations in design variables, the latter of which applies to mapping the performance space (Chen & Yuan, 1998). Applying robust design to design variable uncertainty attempts to reduce the probability of iteration between design teams by finding design and performance sets that are insensitive to the decisions of other design teams (Parkinson, Sorensen, & Pourhassan, 1993; Tsui, Allen, Chen, & Mistree, 1996).

Methods proposed for robust design have shown that developing a performance space mapping involves more than just mapping attainability. Wood and Agogino (2005) use a joint PDF among performance variables to map heterogeneous design spaces, in which some areas of the design space are less achievable than others due to external, uncontrollable factors. Work by Otto and Antonsson (1993) has shown the need to capture constraints in applying probabilistic-based robust design. Additionally, work by Parkinson et al. have highlighted the need to focus on feasibility robustness, wherein a satisfactory design must also be robust to changes in constraints (Parkinson *et al*., 1993). Probabilistic sets have some advantages over interval methods in that multiple probability functions can be created, for example a PDF of attainability can be multiplied by a PDF of performance preference, and then shared between design teams (Eggert & Mayne, 1993).

The probabilistic methods proposed above suffer much of the same classification errors as the interval classification methods because they model each variable independently. Performance variables nearly always have some coupling, which is exhibited by performance tradeoffs, for instance engine weight and power. The Inductive Design Exploration Method (IDEM) creates a multivariate PDF by discretizing the design space into an *n*-dimensional grid, and placing a sample point at each grid intersection (Choi *et al*., 2007). Each sample point is then put through a complete experimental or simulated evaluation. A multivariate feasibility classification PDF is developed by placing an n-dimensional Gaussian kernel on each sample point that meets the feasibility criteria. The IDEM uses a genetic algorithm to generate the next generation of sampling points based on the highest performing design points from the previous generation. The IDEM method facilitates collaborative design by allowing design teams to perform analyses in parallel, but is very computationally expensive due to the number of sample evaluations required to populate the *n*-dimensional grid. The Bayesian network classifiers introduced in the next section provides a significantly more efficient means of creating a multivariate design space PDF, and the work presented in subsequent chapters of this thesis extends this method to the mapping of the performance space.

2.1.4 Bayesian Network Classifiers

The methods described thus far are used to classify a ranged set of independent design variables between design teams. The assumption that variables in engineering systems independently affect the performance of the overall system is instinctively poor,

as engineering problems typically exhibit complex nonlinear relationships. Shahan and Seepersad proposed using Bayesian network classifiers to identify the satisfactory region(s) of each design team's design space (Shahan & Seepersad, 2010). The potential advantages of the Bayesian network classifier design tool are its ability to: (1) capture arbitrarily shaped regions of the design space, (2) combine prior expert knowledge with results from design space sampling, (3) interface with and/or provide a search process for exploring the design space by easily updating the classifier with new sample data, and (4) break from the assumption of variable independence. Bayesian network classifiers have also recently been shown to be effective at classifying a discrete design space (Backlund, 2012), in addition to the continuous spaces studied by Shahan and Seepersad. This section presents an overview of Bayesian network classifiers, and identifies the aspects of this method that will be extended in subsequent chapters of this thesis.

The Bayesian network classifier uses probability distributions for its classification, and is based upon previous research from several groups (Friedman, Geiger, & Goldszmidt, 1997; Hoffmann & Tresp, 1996; John & Langley, 1995; Pérez, Larrañaga, & Inza, 2009). Shahan and Seepersad proposed creating two classes, one for the satisfactory design space and one for the unsatisfactory design space. For an n variable design space, a PDF is created for each class by centering an n-dimensional Gaussian probability distribution on each sample point in the class (John & Langley, 1995; Scott, 1992). The Gaussian probability distributions of each class are aggregated into a weighted sum called a kernel density estimate (KDE) (Shawe-Taylor & Cristianini, 2004). The resulting KDE is used to represent the likelihood that an unknown point in the

design space belongs to the satisfactory or unsatisfactory class. The unknown point is classified according to the class conditional PDF that has a larger value at that unknown point. Points in the design space that have equal probability densities with respect to both KDE's are on the decision boundary between satisfactory and unsatisfactory regions of the design space.

The shape of the performance space is distinctly different than that of the design space, and extending the work of Shahan and Seepersad's Bayesian network mapping of the design space to map the performance space requires several augmentations. The design space typically has upper and lower bounds assigned for each design variable, and therefore a design can theoretically be created by selecting any point from this hyperrectangular space. The shape of the performance space, however, is typically unknown. While each point, x_i, in the design space corresponds to a point, y_i, in the performance space through an objective function $f(x)$, the points in the performance space do not share this same quality in mapping back to the design space. One point in the performance space may correspond to multiple points in the design space or none at all. The subset of the performance space that can be mapped from the design space is the image of the objective function, and it represents the attainable performance space. It is important to map this performance space, especially in cases of multilevel or hierarchical design, in which the output (performance space) of a lower-level problem may become the input (design space) of an upper-level problem, and it is important to identify mutually attainable designs. An example of this type of design process is shown in Figure 2.3, where the small scale performance space corresponds to the large scale design space.

The large scale design team must limit is search space to the attainable and feasible small scale performance space. Defining the attainable performance space using Bayesian network classifiers is the primary challenge addressed in Chapter 3.

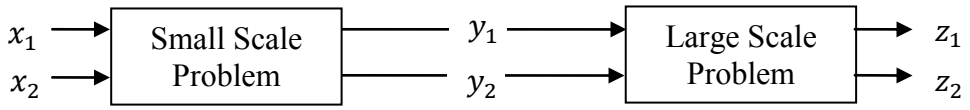

Figure 2.3. An example of a multi-level design problem.

2.2 Quantifying Flexibility

The set-based design paradigm gained notoriety from the evidence presented by Toyota in the 1990's of significantly reduced product lead times (Sobek *et al*., 1999). Studies of Toyota's design practices found that these outcomes were a result of delaying its design decisions longer than its competitors (Ward *et al*., 1995). Specifically, the decisions that Toyota delayed were those that narrowed the size of the design space that was used to search for new and improved designs. As knowledge of the design space is very limited early in the design process, arbitrarily reducing the size of the design space at that point in time may ultimately result in the exclusion of superior designs. Developing a greater breadth of knowledge of the design space also allows designs to be easily adapted upon changes to performance requirements. An effective set-based design approach gradually narrows the design space by developing balance between exploration of large regions, and exploitation of localized regions of the design space (Sobek *et al*.,

1999). In the context of this thesis, delaying design decisions is synonymous with maintaining *design* and *performance flexibility*.

Design flexibility is generally regarded as a measure of the size of the design space that satisfies a given set of performance requirements (Suh, 1990). The size of the satisfactory design space is an indication of the amount of variation design variables can undergo while still meeting the performance requirements. Similarly, performance flexibility is a measure of a design team's ability to produce satisfactory designs even if performance requirements change. Performance flexibility is therefore a measure of the size of the attainable performance space that meets the performance requirements. While numerous methods and metrics have been proposed to quantify design and performance flexibility, an agreed upon method has yet to be developed (Ferguson, Siddiqi, Lewis, & de Weck, 2007). In general, the methods to quantify flexibility are associated with the ability to satisfy a set of performance requirements, and the deviations between quantification methods is primarily a result of varying techniques used to represent performance sets and performance requirements.

2.2.1 Developing Performance Requirements

Developing performance requirements is a nontrivial task, as the requirements represent a designer's subjective assessment of performance preference. An additional challenge in defining performance requirements is incorporating multiple preferences, for instance manufacturing, safety, and cost preferences, into an aggregate preference function. Thurston extended a method developed by von Neumann and Morgenstern to

use utility theory to model preference functions (Thurston, 1991; von Neumann & Morgenstern, 1947). A utility preference function can combine multiple, nonlinear preference functions into an overall measure of a design's worth. A performance requirement using a utility preference function is created by defining a minimum utility value to determine if a design is satisfactory or not, which unfortunately reintroduces subjectivity into the requirements.

Wallace *et al*. (1996) proposed assigning each performance variable a probability distribution measuring a designer's belief that a design will be satisfactory. In a similar approach, fuzzy preference functions have been used to capture a designer's uncertainty in setting performance requirements to specific threshold values by creating a function on each performance variable measuring the belief that each performance value will result in a satisfactory design (Antonsson & Otto, 1995). Fuzzy and probabilistic preference functions share a similar concept, but differ in their effectiveness in propagating their preference functions across design teams in defining system wide design satisfaction. A design process that implements fuzzy or probabilistic preference functions typically takes a negotiation approach, in which design teams seek to maximize the system-wide probability of a design being satisfactory. This type of approach results in a highly iterative point-based design process, but can be converted to a set-based design process by defining a satisfactory probability threshold. For instance, a performance set created from a fuzzy preference function and threshold value of 0.75 identifies the performance values having a preference function of greater than 0.75, and represents the set of attainable performance values having a 75% probability of being satisfactory. Defining

performance requirements by a probability threshold also allows a design team to gradually narrow down their design space by selecting a low initial probability threshold, and slowly raising the threshold until a final, satisfactory design is agreed upon.

While there are many different methods to represent performance preferences, precise satisfactory performance requirements are necessary to be used in set-based design. In the remainder of this thesis it is assumed that design teams have been given performance requirements or thresholds that can be used to classify designs as satisfactory or not.

2.2.2 Flexibility Metrics

In design literature the terms design flexibility and performance flexibility are often interchanged, and nearly always refers to what is called performance flexibility in this thesis. This is presumably due to the focus of design research on developing a range of performance capabilities for mass customization (Jiao & Tseng, 2004) through design paradigms such as product platforms, which develop a product line achieving a range of performances while sharing common design modules (Simpson *et al.*, 2001), and reconfigurable systems, which can adaptively change its physical shape to achieve desired performance under predictable situations (Ferguson & Lewis, 2006). Achieving a range of performance capabilities following Sobek *et al.*'s set-based design guidelines requires intelligent narrowing of the design space, which calls for an accurate metric of design flexibility (Sobek *et al.*, 1999).

One method in the literature that attempts to quantify design flexibility is a metric termed *Information Certainty* (*IC*) (Simpson, Rosen, Allen, & Mistree, 1998). In this metric, Simpson et al. recognize the need to develop a balance between the uncertainties in the final selection of design variables in addition to exploring the achievable system level performance capabilities. *IC* measures the uncertainty of design variable selection, and is defined as the extent to which a design variable value is known precisely at a certain point in the design timeline, as is calculated by Equation 2.1.

$$\mathrm{IC} = \frac{1}{m}\sum_{i=1}^{m}\left(1 - \frac{\Delta x_i}{x_{iu} - x_{il}}\right) \qquad 2.1)$$

Where m is the number of design variables, Δx_i is the range of the i^{th} design variable that produces acceptable performance specifications, and x_{iu} and x_{il}, correspond to the initial upper and lower bounds placed on the i^{th} design variable. The implication of Equation 2.1 is that by the time the design process is finished, a single design point has been chosen, and the *IC* becomes unity. Conversely, when the range of design variables encompasses the entire lower to upper range of the design variable, no decisions have been made to narrow the design space, and the *IC* is 0. By treating each design variable independently, the *IC* metric uses an interval representation of the design space. Representing design variable sets as intervals has inherent classification errors, and can be improved by utilizing Bayesian network classifiers (Shahan & Seepersad, 2012).

In addition to quantifying design flexibility, there have been numerous metrics proposed to quantify performance flexibility. One of the earliest metrics proposed to quantify performance flexibility is Suh's Information Axiom (Suh, 1990). Suh defines a

metric *Information Content* (*I*) that quantifies the relationship between the range of achievable performances and range of performances that satisfy system performance requirements. The *I* metric is presented in Equation 2.2, and is intended to quantify the probability that a set of designs will satisfy a set of performance requirements.

$$\mathrm{I} = \log\left(\frac{\text{system range}}{\text{common range}}\right) \qquad (2.2)$$

The system range in Equation 2.2 is the achievable performance range that the set of designs can achieve, and the common range is defined as the intersection range of the achievable performance range and the target performance range. In this metric, the case where there is no common range between the achievable and target ranges, *I* is equal to infinity and there is no knowledge of how to create an acceptable design. At the other extreme when there is complete overlap between the achievable performance range and the target range, *I* is at a minimum and is equal to 0. Suh believed that by minimizing *I* a design team could focus on simplicity by searching for designs having minimal information content. Additionally, Suh's quantification metric treats each performance variable independently and suffers the classification inaccuracy inherent to intervals.

Suh's information content metric was modified by Simpson *et al*. to attempt to quantify performance flexibility on a range from 0 to 1 (Simpson *et al*., 1998). The modified information content metric was termed *Design Freedom* (*DF*) by Simpson et al, and quantified the amount a system can change while still maintaining acceptable performance metrics (Simpson *et al*., 1998). The metric for *DF* is given in Equation 2.3.

$$DF = \sum_{i=1}^{n} \left(\frac{TR_i \cap PR_i}{PR_{i,initial}} \right) \quad (2.3)$$

Where n is the number of performance specifications, TR_i is the target range of the i^{th} performance specification, and PR_i is the performance range which is attainable as well as satisfactory. From this definition, *DF* is equal to 1 at the start of the design process and then subsequently decreases towards 0 as the satisfactory design space is narrowed. The novelty in Simpson *et al.*'s *DF* metric is that it can be combined with their *IC* metric to intelligently narrow the design space (Panchal et al., 2007). However, both the *IC* and *DF* metrics utilize interval representations of the design and performance spaces, and suffer classification errors that could potentially lead to a design process that does not converge to a satisfactory solution.

In a similar performance flexibility metric to *DF*, Chen *et al*. proposes *Design Capability Indices* (*DCI*) that use a probabilistic representation of the attainable performance set (W. Chen, Simpson, Allen, & Mistree, 1996). This method assigns a Gaussian PDF to the value each performance variable will take by calculating the mean and variance of each performance variable within the selected performance set. The *DCI* is then equal to the difference between the mean of the PDF and the lower and upper bounds of the performance requirements, divided by a maximum defect rate represented by a standard deviation. Chen and Yuan combine *DCI* with a performance utility preference function to form a *Design Preference Index* (*DPI*) (W. Chen & Yuan, 1998). The *DPI* metric allows a design team to determine if a design is considered feasible, based on whether or not the manufacturing tolerances are greater than the variance in the

design variable set. The limitations of *DCI* and *DPI* come in the approximation of each performance variable's PDF, wherein the PDF is used to model performance uncertainty rather than performance capability. These probabilistic-based set representation methods are therefore more tailored for use in robust design approaches, rather than for mapping the performance space in a set-based design approach.

Olewnik and Lewis proposed a more abstract metric of performance flexibility that analyzed a design set's performance Pareto frontier, rather than the entire attainable performance space (Olewnik & Lewis, 2006). A Pareto set occurs when there are multiple competing performance variables; for any member of the Pareto set, performance can be improved with respect to one performance variable only by degrading another performance variable (Pareto, 1971). Pareto sets are therefore very useful in analyzing the tradeoffs between design concepts, and have been used in the selection of design concepts having dissimilar design variables (Mattson & Messac, 2003). The performance flexibility metric proposed by Olewnik and Lewis is defined as the "distance" between the extreme points of the Pareto performance set, or the sum of pair-wise distances in cases with more than two performance variables (Olewnik & Lewis, 2006). The extreme points in a Pareto set represent the most preferred design with respect to each performance variable, and the distance between them represents the performance variability that can occur within a set of designs. The basic idea of using the extreme Pareto point performance flexibility metric is that as the design space is narrowed the flexibility also decreases to zero. However, this method ignores the non-Pareto set performance points, and thus quantifies only one aspect of performance

flexibility. Furthermore, this flexibility metric is not applicable in a multilevel design problem, in which a high-level design team may choose non-Pareto points from a low-level design team if they improve high-level performance.

In an effort to determine the potential of finding a satisfactory solution within a design space, Clevenger and Haymaker adopt a frequentist approach to approximating performance flexibility (Clevenger & Haymaker, 2011). In this metric, performance flexibility is approximated as the ratio of the number of satisfactory sample points to unsatisfactory sample points. Although this method requires an extremely high number of sample points to achieve an accurate performance flexibility approximation, it may in fact be a better approximation of performance flexibility than using interval set representations. The method proposed in this thesis to quantify performance flexibility is relatively similar to this frequentist method, but incorporates a method to infer performance attainability between known sample points. Therefore, the method proposed in this thesis will be able to achieve higher classification accuracy in fewer sample point evaluations than this approach.

2.3 Discussion

A set-based design paradigm is a relatively new concept, and the methods reviewed in this chapter lack the capability to be employed in a complex design problem. Prior methods of set representation have inherent error associated with them, and are therefore not suitable to be used in practice. The recent Bayesian network classifier method to map the design space has been shown to vastly outperform the interval and

probabilistic representations of the design space, but has not yet been applied to the performance space. Chapter 3 will address this need by introducing extensions of the Bayesian network classifier method necessary to map the performance space.

An effective set-based design process also requires an accurate approximation of design and performance flexibility, which has yet to be proposed. The quantification accuracy of design and performance flexibility is reliant on the method of set representation. A method for the quantification of design and performance flexibility is introduced in Chapter 4, which uses a Bayesian network classifier for its set representation. The combination of an accurate representation of the performance space and the quantification of design and performance flexibility will be the focus of the remainder of this research. These methods will then be demonstrated on a hierarchical material design problem in Chapter 5.

Chapter 3: Representing and Classifying the Performance Space

In a single-level design problem it is only necessary to map the satisfactory design space, which corresponds to the set of design variable values that meet a set of performance requirements. Typically in a single-level design problem, the design space is constrained by placing lower and upper bounds on each design variable to create a hyperrectangular space. In a multi-level design process, however, the design space can be arbitrarily shaped, as a result of the design space for an upper-level design team corresponding to the feasible performance space for a lower-level design team. A two-level design process is illustrated in Figure 3.1, displaying how the large-scale design team's design space is irregularly shaped as a consequence of it being defined by the feasible small-scale performance space. Therefore, in a multi-level design process it is necessary to create a mapping of the feasible performance space, in addition to the satisfactory design space. A multi-level mapping of each design team's design and performance spaces facilitates an effective concurrent collaborative design by allowing design teams to share their feasible sets of designs, which can be intersected to find a mutually feasible design.

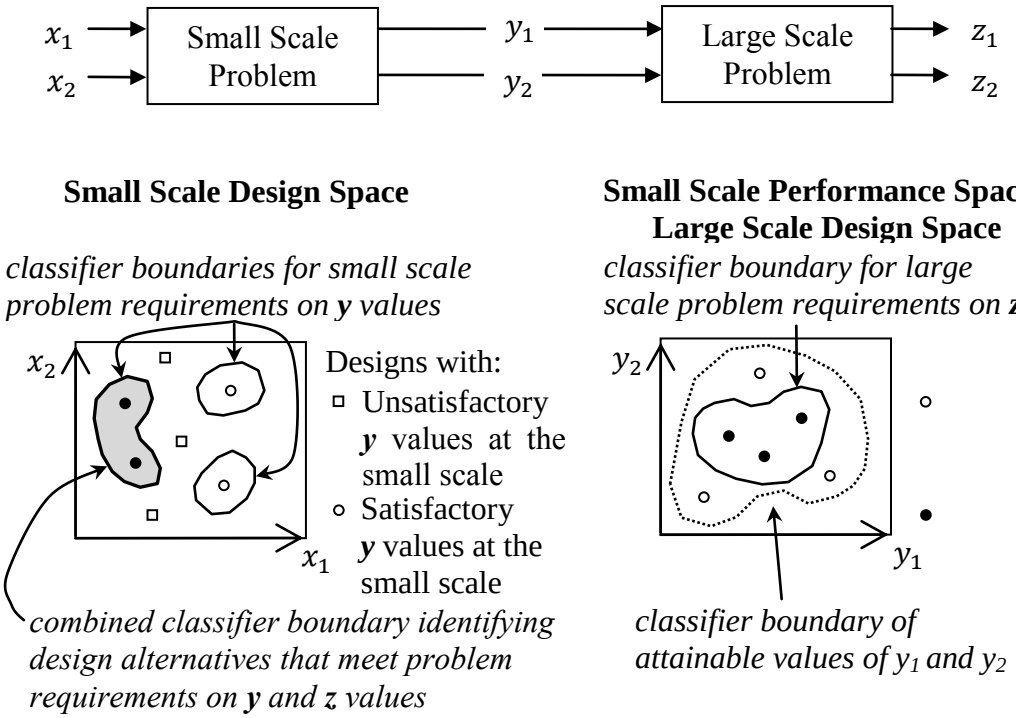

Figure 3.1. The backward propagation of requirements across levels of a multi-level design problem.

The performance space can be classified into a hierarchy of subspaces, as depicted in Figure 3.2. The attainable performance space is a subspace of the overall performance space, representing all possible function mappings from the design space. The existence of constraints in the design process, such as manufacturing tolerances, divides the attainable performance space into a feasible performance space and an infeasible performance space. The performance point of a design is classified as feasible if it does not violate any constraints, and infeasible otherwise. The feasible performance space can also be then divided into two subspaces based on meeting a set of performance requirements. The set of feasible sets of designs in the performance space that meet all of

the performance requirements are classified as satisfactory, and those that do not are classified as unsatisfactory.

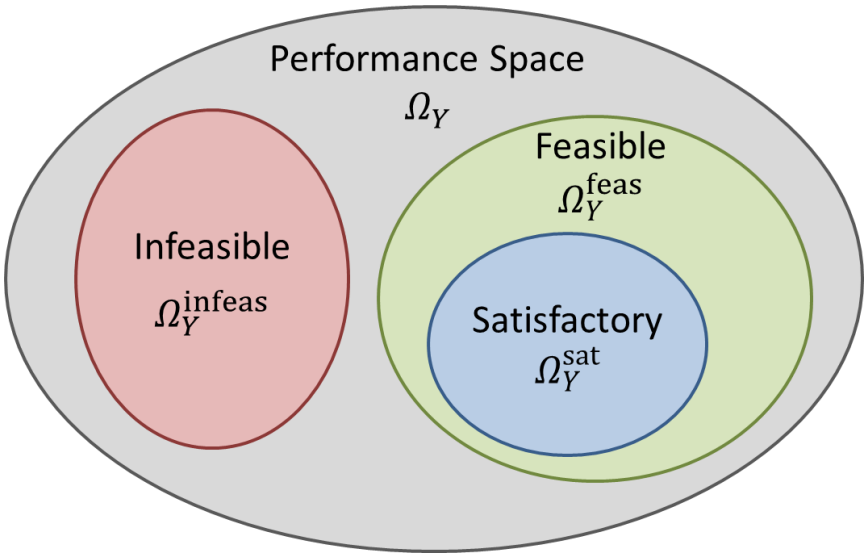

Figure 3.2. Illustration of the classifications in a generalized performance space

In a generalized design problem, a designer explores the design space in search of a design that corresponds to a point in the performance space that meets the designer's performance requirements. Using the nomenclature defined in Table 3.1, a design can be mapped to a corresponding performance, represented as a vector of m performance variables, $\boldsymbol{y} = [y^1, y^2, \ldots, y^m] \in \Omega_Y^{\text{att}} \in \Re^m$, corresponding to a unique point in the m-dimensional, attainable performance space. A function that maps points in the design space to points in the attainable performance space, $\Omega_X \rightarrow \Omega_Y^{\text{att}}$, typically represents a computational simulation or physical experiment.

Table 3.1 Nomenclature	
m	Number of performance variables
N	Number of training points
$\boldsymbol{x}$	A design instance, where $\boldsymbol{x} = [x^1, x^2, \dots, x^n]$
$\boldsymbol{y}$	A performance instance, where $\boldsymbol{y} = [y^1, y^2, \dots, y^m]$
$\boldsymbol{y}_{req}$	Set of performance requirements
c_k	Classification category k
Ω_X	Design space
$\Omega_Y^{c_k}$	Performance space belonging to class k

The classifier introduced in this chapter is based on previous research in Bayesian networks (Pearl, 1988), and uses kernel density estimation techniques (Fukunaga, n.d.; John & Langley, 1995; Katkovnik & Shmulevich, 2002; Ledl, 2002; Parzen, 1962; Scott, 1992; Shawe-Taylor & Cristianini, 2004) to produce a probability density function(s) from a set of designs in the performance space of known classification, called training points. Developing the kernel-based BNC allows a designer to classify an arbitrarily shaped satisfactory performance space, and update it easily with additional training points (John & Langley, 1995; Pérez, Larrañaga, & Inza, 2009). The proposed classifier

specifically extends the work of Shahan and Seepersad's BNC mapping of the design space (Shahan & Seepersad, 2012), by adding necessary augmentations to facilitate a BNC mapping of the performance space. Shahan and Seepersad's design space BNC sought to classify the satisfactory design space, corresponding to designs that meet all performance requirements. In the performance space, however, determining if a performance point is satisfactory or not is trivial, as the performance requirements are explicitly given in terms of the performance space. The challenge in classifying the performance space is to: (1) distinguish the *attainable* performance space from the *unattainable* performance space, and (2) distinguish the *feasible* performance space from the *infeasible* performance space.

A background of Bayesian decision theory and the mathematical formulation of the proposed BNC are presented in Section 3.1. The BNC produces a probability distribution that is used to extrapolate knowledge between training points, and is employed in Section 3.2 to define the boundary of the feasible versus infeasible regions of the performance space. The BNC will then be used in Section 3.3 to classify the attainable performance space versus the unattainable performance space. The goal of the proposed classifier is to map a feasible performance space of arbitrary shape that converges in classification accuracy as additional training points are introduced.

3.1 The Bayesian Performance Space Classifier

Evaluating a design concept, through experimentation or computational simulation, gives designers knowledge of the corresponding performance value for each

evaluated concept. The known sets of designs in the performance space can each be assigned a categorical classification, based on meeting performance requirements and design constraints. The proposed classifier aids designers' mapping of the performance space by replacing intuition with a more robust mathematical formulation. The classifier is given a finite set of known designs in the performance space, each represented in an *m*-dimensional vector $\boldsymbol{y}$, and outputs one probability density function for each defined class, to be used to classify unknown points.

The set of training points can then be used to create a class conditional probability of a performance point given a class, expressed as $p(\boldsymbol{y}|c)$. The conditional probability of the class given a performance point, $p(c|\boldsymbol{y})$, however, is the desired conditional probability used to predict the classification of a performance point of unknown class. Bayes formula is used to transform the unknown *posterior* probability, $p(c|\boldsymbol{y})$, into an expression involving the known *likelihood*, $p(\boldsymbol{y}|c)$, according to Equation 3.1.

$$P(c|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|c)P(c)}{p(\boldsymbol{y})} \tag{3.1}$$

The denominator of Equation 3.1 represents the *evidence* factor, and is essentially a normalization scale factor used to ensure the posterior probabilities from all classes sum to 1. The *evidence* factor is found by summing the numerator of the right side of Equation 3.1 for each class, according to Equation 3.2.

$$p(\boldsymbol{y}) = \sum_{i=1}^{k} p(\boldsymbol{y}|c_i)P(c_i) \tag{3.2}$$

The *prior* probability, $P(c)$, is approximated as the ratio of training points belonging to each class, as shown in Equation 3.3 where N is the total number of training points, and N_c is total the number of training points belonging to class c. The frequency of occurrence of each class is given a "padding" of one occurrence in order to improve the approximation accuracy for classes with low sample sizes, by smoothing the changes of the approximation as N_c increases.

$$P(c) \cong \frac{N_c + 1}{N + 2} \tag{3.3}$$

Classifying an unknown performance instance, $\boldsymbol{y}$, is performed according to *Bayes decision rule*, by finding the class that has the largest class conditional posterior probability, $P(c|\boldsymbol{y})$, according to Equation 3.4.

$$\text{Decide } c_i \text{ if } P(c_i|\boldsymbol{y}) > P(c_j|\boldsymbol{y}) \text{ for } j = 1 \ldots m,\ j \neq i \tag{3.4}$$

The decision rule in Equation 3.4 is demonstrated in Figure 3.3, showing the conditional probability of the class given a performance point, for a classifier having two classes in a univariate performance space.

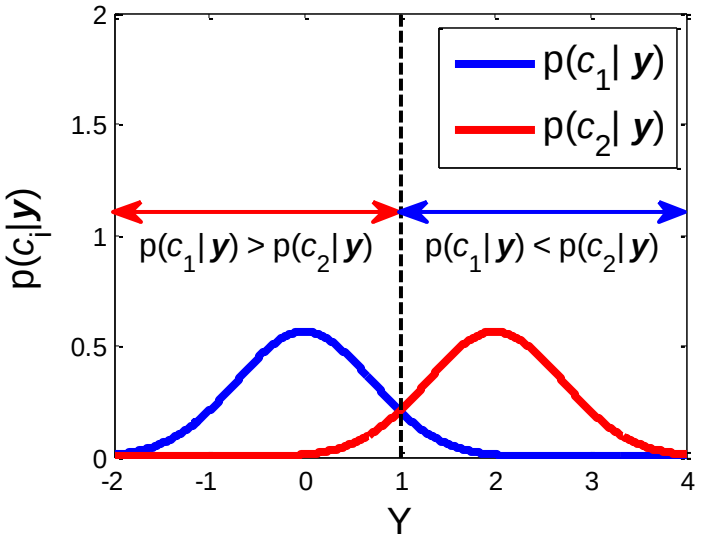

Figure 3.3. Demonstration of Bayes decision rule on classifying a univariate performance space

The classification decision is simplified by neglecting the *evidence* scale factor, $p(\boldsymbol{y})$, and is shown for the two class case in Equation 3.5. A loss factor, λ_{ij}, describing the risk in misclassifying a performance point as belonging to class c_i when it actually belongs to class c_j can be incorporated into the decision rule of Equation 3.4, according to Equation 3.5. The loss factor can be introduced to reduce the misclassification error, which can be approximated by cross-validating the classifier, by precluding training points from the classifier and subsequently determining if the excluded points were accurately classified or not. If there is insufficient data to approximate the loss factor, or if it is believed that the error is symmetric across all classes, the loss factors can all be set to 1.

$$\text{Decide } c_1 \text{ if } \lambda_{21}\, p(\boldsymbol{y}|c_1)P(c_1) > \lambda_{12} p(\boldsymbol{y}|c_2)P(c_2) \tag{3.5}$$

The decision boundary between class c_1 and c_2 occurs when both sides of Equation 3.5 are equal. The loss factors have the effect of scaling the posterior probabilities of Equation 3.4 by ratio of loss factors between classes, thereby extending the classification boundary. The following section provides the mathematical framework to translate a training set into a likelihood probability distribution, $p(\boldsymbol{y}|c)$.

3.2 Capturing the Knowledge from Training Points

Designers achieve a better understanding of the design and performance space by evaluating the performance of new design concepts. With each concept evaluation, a challenge to designers is to capture and quantify this acquisition of new information. The proposed BNC uses the concept evaluations as a training set to determine the conditional probability of each performance variable given the class label, $p(y_i|c)$. Then in order to determine the overall likelihood function, $p(\boldsymbol{y}|c)$, the conditional dependence between performance variables given the class must be specified. Performance variable y_i is defined to be conditionally *dependent* on y_j given the class, if Equation 3.6 holds true for all values of y_i, y_j, and c, and $p(c) > 0$. The performance variables y_i and y_j are defined to be conditionally *independent* given the class if Equation 3.6 is not true.

$$p(y_i|y_j, c) \neq p(y_i|c) \tag{3.6}$$

The conditional dependence between performance variables is typically represented as directed acyclic graphs (DAG), as seen in Figure 3.4 (Pearl, 1988). In DAG's, nodes correspond to a unique performance variable, and arrows correspond to conditional dependence between variables. A node with a departing arrow is defined to be the *parent* of the *child* node at which the arrow ends. A performance variable y_i is dependent on the set of its parent variables, represented as $\boldsymbol{pa}_i$, and independent of all non-descendent performance variables.

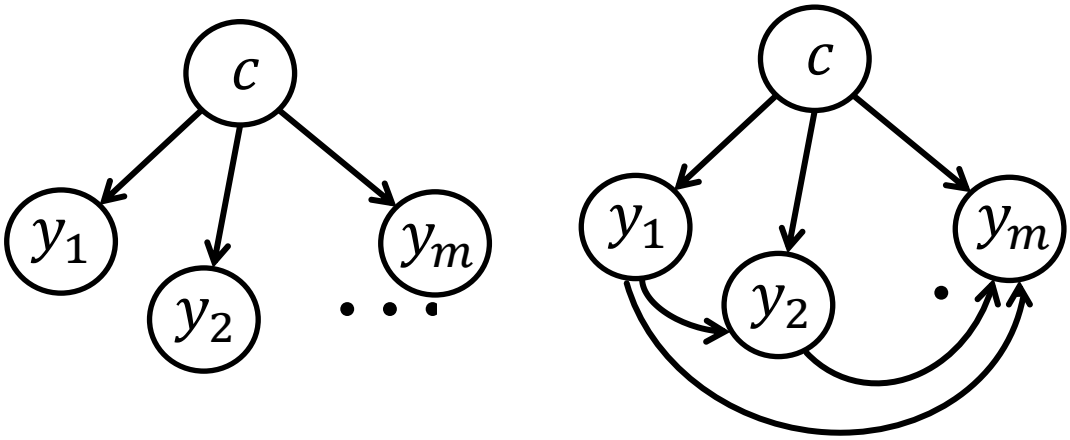

Figure 3.4 Directed Acyclic Graph of a Fully Independent (left) and Fully Dependent (right) networks.

From the creation of a DAG model of the dependence between performance variables, an overall joint probability distribution can be calculated according to Equation 3.7.

$$p(\boldsymbol{y}) = \prod_{i=1}^{m} p(y_i|\boldsymbol{pa}_i) \quad (3.7)$$

The two extremes of variable dependencies are illustrated by the DAG's in Figure 3.4. The fully dependent network occurs when each performance variable, y_i, is conditionally dependent on the performance variables having a lower ordinal number, y_1, … , y_{i-1}. The fully independent network is a network where each performance variable is conditionally independent of all other performance variables given the class. The fully independent network is also referred to as a Naïve Bayes classifier due to its strong independence assumption, and has been seen to yield surprisingly accurate classification, even in networks having inherent dependence between variables (Friedman, Geiger, & Goldszmidt, 1997). Additionally, the fully independent network allows for the simplification of Equation 3.7 into the form shown in Equation 3.8. For these reasons, the networks examined in this thesis are all modeled as fully independent networks.

$$p(\boldsymbol{y}) = p(y_1)p(y_2)\dots p(y_m) = \prod_{i=1}^{m} p(y_i) \quad (3.8)$$

A kernel density estimate (KDE) technique is used to calculate the conditional probabilities at each node of the Bayesian network (Parzen, 1962; Shahan & Seepersad, 2012). Kernel density estimation is a nonparametric technique for estimating the probability density function, relying only on the values in the training set, and can therefore be generically applied and tuned to unique design problems without needing to change the underlying algorithm. KDE's are also a computationally efficient density estimation technique, requiring only a polynomial increase in computational resources for an exponential increase in dimensionality (Shawe-Taylor & Cristianini, 2004). The KDE places an *m*-dimensional kernel probability distribution at each training point, which

when summed forms a continuous probability density function over the performance space. The basic equation for a KDE evaluated at a point $\boldsymbol{y}$ in the performance space is given in Equation 3.9 (Rosenblatt, 1956).

$$p(\boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{K}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}^i, \boldsymbol{\theta}\right) \quad (3.9)$$

where N represents the number of training points, $\hat{\boldsymbol{y}}^i$ is an $m \times 1$ vector of the ith training point, and $\mathcal{K}$ is the kernel function. The value returned by the kernel function is dependent on the geometric distance between the performance point of interest, $\boldsymbol{y}$, and each training point, $\hat{\boldsymbol{y}}^i$, as well as a set of kernel parameters, $\boldsymbol{\theta}$. For the methods presented in this thesis, a Gaussian kernel function is used, which has only one kernel parameter: the covariance matrix, $\boldsymbol{H}$. The Gaussian distribution function is shown in Equation 3.10, where the kernel mean vector, $\boldsymbol{\mu}$, is centered on training point, $\hat{\boldsymbol{y}}^i$, in Equation 3.9.

$$\mathcal{K}(\boldsymbol{y}, \boldsymbol{\mu}, \boldsymbol{H}) = \frac{1}{\sqrt{(2\pi)^m |\boldsymbol{H}|}} e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T \boldsymbol{H}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})} \quad (3.10)$$

Assuming a diagonal covariance matrix, the entries of the covariance matrix, $\boldsymbol{H}$, are determined by Equations 3.11 and 3.12.

$$\boldsymbol{H}_{ii} = {h_i}^2 \quad (3.11)$$

$$\boldsymbol{H}_{ij} = 0 \text{ , where } i \neq j \quad (3.12)$$

3.12)

Where h_i is the standard deviation of the kernel function along the i^{th} dimension. The kernel function's standard deviation is also referred to as the *kernel bandwidth* in literature, and will be referred to as such in the remainder of this thesis in order to clearly differentiate the kernel's standard deviation from the standard deviation of the training set. The kernel bandwidth serves as a smoothing parameter to adjust the region of influence of the training points on the unsampled space around them. The effect of the kernel bandwidth on the overall KDE is demonstrated in Figure 3.5. In this example, a training set of 10 points has been sampled from a univariate normal distribution having a mean of 0 and a standard deviation of 1. Low values for the kernel bandwidth results in a noisy KDE that has distinct characteristics from each individual training point. Conversely, high values for the kernel bandwidth results in smoothed out KDE that combines the influence of each training point to give a single peaked distribution.

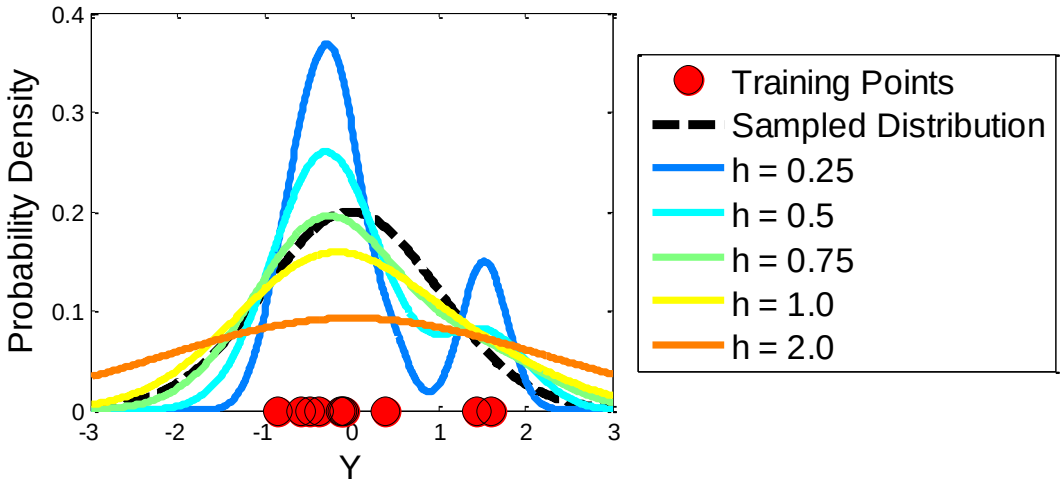

Figure 3.5. Kernel density estimates for several values of kernel bandwidth.

The kernel probability distributions for a single training point from Figure 3.5 are shown in Figure 3.6. A kernel with a low bandwidth is seen to have a high probability peak directly over the training point that diminishes quickly as the distance from the training point is increased. In the limit that the kernel bandwidth approaches zero the resulting KDE will be zero for all values except for the training point values. Conversely, in the limit that the kernel bandwidth approaches infinity, the resulting KDE will be constant over the entire performance space. Clearly, neither extreme case of the kernel bandwidth will provide beneficial insights to designers.

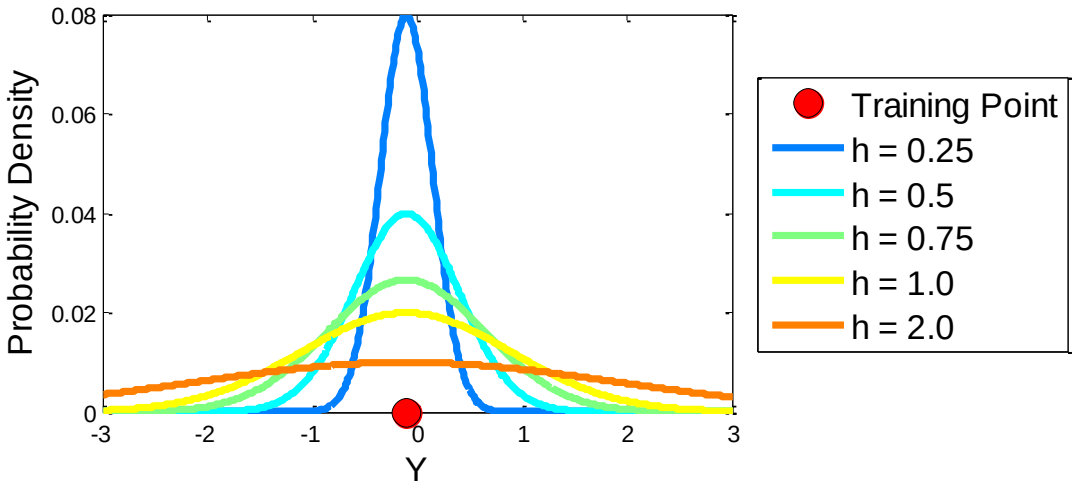

Figure 3.6. Kernel probability distributions of one training point with several values of kernel bandwidth.

The kernel bandwidth parameter should be chosen so that the resulting KDE accurately captures the underlying distribution of the performance space. In practice, however, the underlying distribution is not known. The KDE should therefore capture the designer's knowledge of the performance space, which is contained in the set of training

points. In the example shown in Figure 3.5, the set of training points are drawn from the same distribution, but they have two clusters: one primary cluster around -0.5 and a secondary cluster around 1.5. The resulting KDE should therefore retain this characteristic, and not necessarily match the underlying distribution from which the training points were sampled. As more samples are generated, the KDE should approximate the underlying distribution more precisely. Heuristics for setting the kernel bandwidth are discussed in depth in Section 3.3.

Following the assumption of a diagonal covariance matrix, the kernel equation from Equation 3.10 simplifies to Equation 3.13, which replaces the center point of the kernel, $\boldsymbol{\mu}$, with the value of the i^{th} training point, $\hat{\boldsymbol{y}}^i$.

$$\mathcal{K}(\boldsymbol{y}, \hat{\boldsymbol{y}}^i, \boldsymbol{h}) = \frac{1}{\sqrt{2\pi}} \prod_{j=1}^{m} \frac{1}{h_j} e^{-\frac{1}{2}\left(\frac{y_j - \hat{y}_j{}^i}{h_j{}^2}\right)} \tag{3.13}$$

The KDE equation from Equation 3.9 can now be written in the full form shown in Equation 3.14.

$$p(\boldsymbol{y}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{K}(\boldsymbol{y}, \hat{\boldsymbol{y}}^i, \boldsymbol{h}) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{\sqrt{2\pi}} \prod_{j=1}^{m} \frac{1}{h_j} e^{-\frac{1}{2}\left(\frac{y_j - \hat{y}_j{}^i}{h_j{}^2}\right)} \right) \tag{3.14}$$

The KDE in Equations 3.9 and 3.14 weights each training point equally, however a weighted average function can be utilized to assign unique weights to each training point, subject to the constraint given in Equation 3.16. The weighted average KDE equation is given in Equation 3.15, where w^i represents the weight assigned to the i^{th} training point.

$$p(\boldsymbol{y}) = \sum_{i=1}^{N} w^i \mathcal{K}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}^i, \boldsymbol{h}\right) = \sum_{i=1}^{N} \frac{w^i}{\sqrt{2\pi}} \prod_{j=1}^{m} \frac{1}{h_j} e^{-\frac{1}{2}\left(\frac{y_j - \hat{y}_j{}^i}{h_j{}^2}\right)} \quad (3.15)$$

$$\sum_{i=1}^{N} w^i = 1 \quad (3.16)$$

The class conditional probability of a point in the performance space, $p(\boldsymbol{y}|c)$, can be calculated using the KDE technique given in Equation 3.15, for each class. A KDE is generated for a given class, according to Equation 3.17, from the set of training points belonging to that class.

$$p(\boldsymbol{y}|c) = \sum_{i=1}^{N_c} w_c^i \mathcal{K}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}^i, \boldsymbol{h}\right) \quad (3.17)$$

Where N_c is the number of training points belonging to class c, and w_c is an array of weightings specific to class c and follows the constraint of Equation 3.16. In the following section, the likelihood function is used to classify the feasible performance space and distinguish it from the infeasible performance space.

3.3 Distinguishing the Feasible Performance Space

As designers build up knowledge of the performance space, through sampling and evaluating points in the design space, the BNC outlined in the previous section can then be used to compile and summarize this information into a useful form. This section discusses how the BNC is used to classify the feasible performance space. Using a two-class system, c_1 represents the feasible region, and c_2 represents the infeasible region of the performance space. A point in the performance space is classified as belonging to

either c_1 or c_2 according to the Bayesian decision rule in Equation 3.4 and 3.5, using the class conditional probability distribution evaluated according to Equation 3.17.

The classification accuracy of the BNC is largely dependent on how well the probability distributions that are generated by the KDE capture the space's underlying distribution. The kernel bandwidth smoothing parameter allows the KDE technique to be tuned to improve the density estimation, and is regarded by researchers in the field as the most influential parameter on the KDE's accuracy (Scott, 1992). Methods to set the kernel bandwidth can be categorized into adaptive methods and heuristic-based methods. Adaptive methods to set the kernel bandwidth are typically formulated as an optimization problem to minimize the classification error, found through cross-validation. In general, adaptively setting the kernel bandwidth yields improved classification accuracy over heuristic-based methods, but can also be significantly more computationally expensive (Dudda, Hart, & Stork, 2001; Fukunaga, 1990; Scott, 1992). Additionally, adaptive methods of setting the kernel bandwidth have been studied quite extensively in literature, as summarized in (Jones, Marron, & Sheather, 1996), and are therefore not investigated in this research. Several heuristics have been proposed for setting the kernel bandwidth, and are desirable because they require minimal computational expense and have been shown to perform very well (Shahan & Seepersad, 2010a).

The Normal reference rule has been introduced as a method to set the kernel bandwidth, and is derived by minimizing the asymptotic mean integrated square error (AMISE) of a multivariate Gaussian distribution using Gaussian kernel functions (Scott, 1992; Silverman, 1986). The normal reference rule is given in Equation 3.18, and is seen

to be a function of the number of dimensions of the performance space, m, the number of training points belonging to class c, N_c, and the standard deviation of all training points in the i^{th} dimension, $\hat{\sigma}_i$.

$$h_{i,c} = \left(\frac{4}{m+4}\right)^{1/(m+4)} \frac{\hat{\sigma}_i}{N_c^{\,1/(m+4)}} \quad (3.18)$$

The first part of Equation 3.18 only depends on the number of dimensions of the performance space, and acts as a constant for the normal reference rule. This constant increases asymptotically to 1 as m goes to infinity, and is never less than 0.924 (Scott, 1992). Approximating this constant as equal to 1 leads to Scott's rule for setting the kernel bandwidth, according to Equation 3.19.

$$h_{i,c} = \frac{\hat{\sigma}_i}{N_c^{\,1/(m+4)}} \quad (3.19)$$

The Normal reference rule and Scott's rule are both derived from maximizing the KDE's accuracy in approximating a Gaussian distribution; however, the problems encountered in engineering design rarely follow such a "regular" distribution. Shahan and Seepersad found that the Normal reference rule and Scott's rule did not reduce the kernel bandwidth fast enough with respect to the number of training points (Shahan & Seepersad, 2010b). A similar heuristic for setting the kernel bandwidth, proposed by John and Langley, was found by Shahan and Seepersad to perform better than Scott's rule, and is given in Equation 3.20. In the heuristic proposed by John and Langley, the scaling

constant α is set to 1, while Shahan and Seepersad found better results from setting α to 0.4.

$$h_{i,c} = \frac{\alpha_c}{\sqrt{N_c}} \quad (3.20)$$

The heuristic given in Equation 3.20 for setting the kernel bandwidth will be used in this research, with several modifications. Equation 3.20 assumes that the performance space has been scaled by the range of each dimension, $y_{i,max} - y_{i,min}$. The range of values in the performance space, however, is typically unknown, and it is therefore more reasonable to scale each dimension by the standard deviation of the training points along that dimension. The standard deviation can be related to the range, by a scaling factor representing the ratio between the standard deviation of a uniform distribution to the standard deviation of a normal distribution, as shown in Equation 3.21.

There are several errors that may be encountered with using this approach for setting the kernel bandwidth. If one class contains a disproportionately high number of training points, Equation 3.20 will define the kernel bandwidth for the class with a high number of training points to be smaller than the other class. Noting that larger kernel bandwidths have a smaller maximum probability, the KDE of the class with fewer training points may become dwarfed by the KDE of the other class. Additionally, having a different kernel bandwidth for each class can lead to classification errors occurring in the tail ends of the probability distributions, due to a larger kernel bandwidth having a smaller rate of decay as the distance from the kernel's mean is increased. Equation 3.21 modifies the heuristic shown in Equation 3.20 to alleviate these issues, and will be used

in the remainder of this thesis. The parameter α in Equation 3.21 is a tuning parameter to increase the classification accuracy of the BNC.

$$h_i = \frac{\sqrt{12}\hat{\sigma}_i \alpha}{\sqrt{N}} \quad (3.21)$$

To demonstrate the BNC's classification of the feasible and infeasible performance spaces, a simple spring design problem is used. The design problem is described in Appendix A, and is taken from (Juvinall & Marshak, 2000; Shahan & Seepersad, 2012). The spring design problem has two design variables, two performance variables, a performance target for each performance variable, and two constraints, making it a simple yet thorough demonstration problem. This example is used through the remainder of Chapter 3 and Chapter 4 to demonstrate the methods presented in this thesis. The Matlab® code used to implement this method is included in Appendix B.

Training points are selected from the design space based on the pseudo-random Halton sequence (Freeman & Halton, 1951), and are then evaluated to determine if they are feasible or infeasible. The results from selecting and evaluating 50 training points are shown plotted in the design and performance spaces in Figure 3.7. A comparison of the design and performance spaces in Figure 3.7 shows that a relatively uniformly distributed set of points in the design space translates to an irregular distribution of points in the performance space.

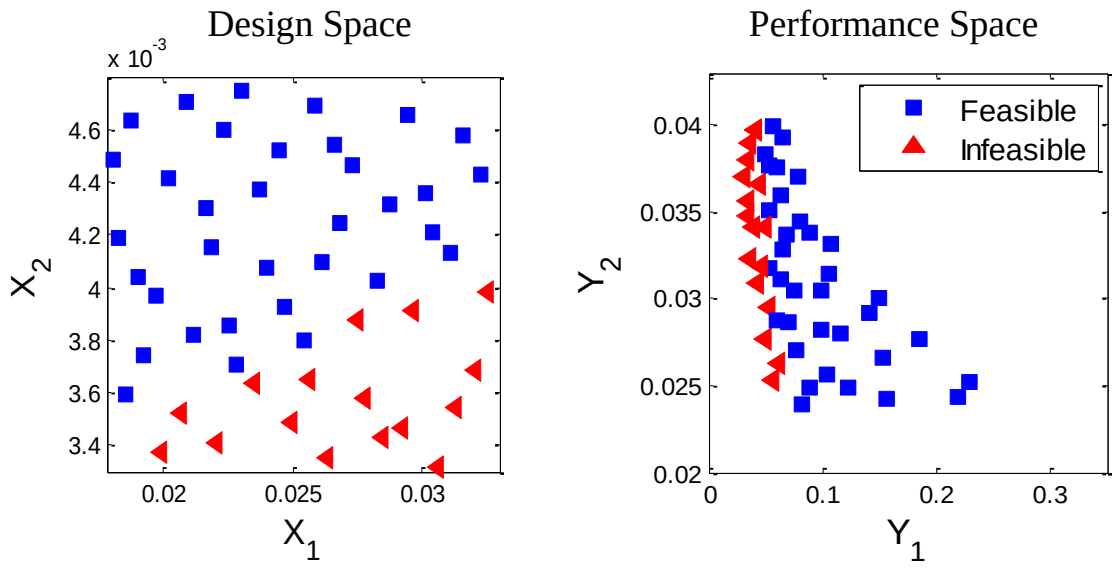

Figure 3.7. The design space (left) and performance space (right) showing 50 training points classified by feasible (blue square) and infeasible (red triangle).

Once a set of training points have been evaluated the class conditional probability distribution, $p(\boldsymbol{y}|c_i)$, is determined for each class according to Equation 3.17, and the prior probability, $P(c_i)$, is calculated for each class according to Equation 3.3. From the likelihood and prior probabilities, the BNC can classify the performance space according to Equation 3.5. The resulting probability distributions for each class are plotted in Figure 3.8 with α set to 0.4.

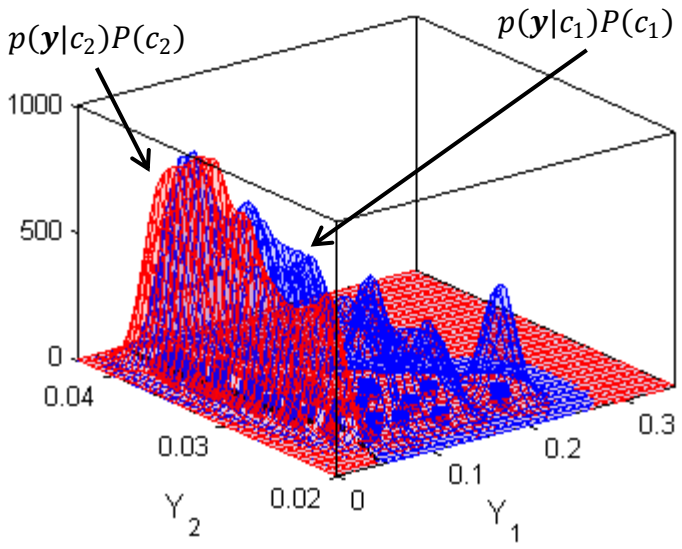

Figure 3.8. The posterior probability distributions for the feasible (blue) and infeasible (red) classes.

The decision boundary separating the feasible and infeasible classes occurs where two probability distributions are equal in Figure 3.8. Alternatively for the two-class BNC, a single normalized decision surface can be determined from the difference of the posterior probability distributions of the feasible and infeasible classes. The difference of the class posterior probability distributions is plotted in Figure 3.9, with a class decision boundary where the z-axis is equal to 0. The BNC mapping of the feasible and infeasible performance spaces is shown in a top-down, 2-D view in Figure 3.10. The decision boundary drawn as a solid black line in Figure 3.10 separates the infeasible performance region to the left from the feasible performance region on the right.

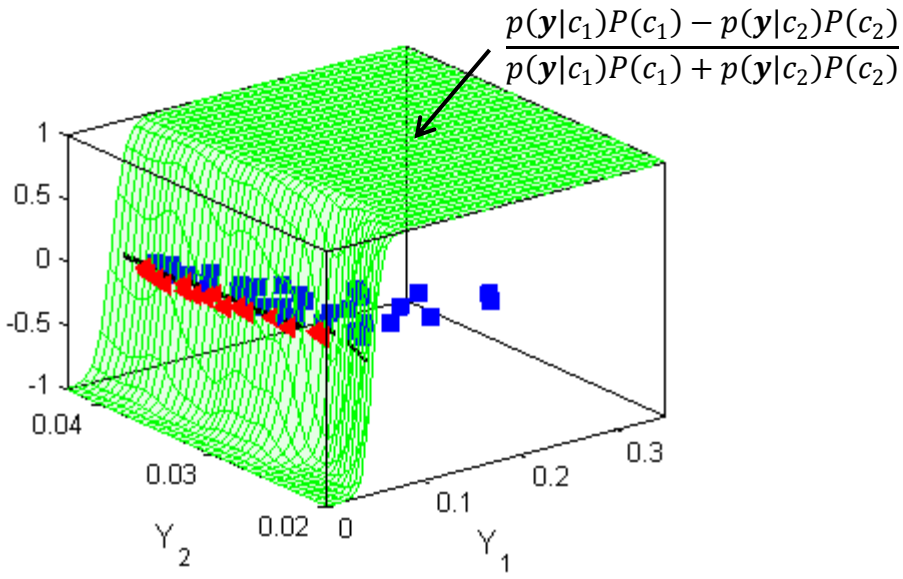

Figure 3.9. The normalized posterior probability decision surface.

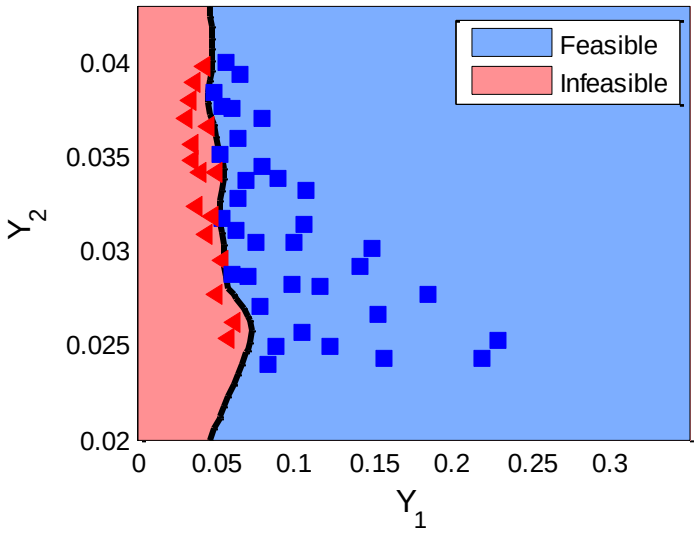

Figure 3.10. A 2-D mapping of the feasible performance space.

To test the classifier's accuracy, a set of known performance values and corresponding class distinctions is evaluated by the BNC. A performance point that is

classified by the BNC as feasible but is actually infeasible represents a false feasible classification. Similarly, a performance point that is classified by the BNC as infeasible but is actually feasible represents a false infeasible classification. The classifier's error rate is defined to be the fraction of false feasible/infeasible points to the total number of evaluated performance points. A test set of 1,000 points, drawn from the Halton sequence, is sampled from the design space and evaluated to examine the BNC's accuracy. Since the classifier's training points were also drawn from the Halton sequence, points in the test set that also appeared in the classifier's training set were removed from the test set. The false feasible and false infeasible error rates of the BNC having 50 training points are plotted as a function of α in Figure 3.11. For values of α less than 1, the error rates of both the false feasible and infeasible classifications are about 5%. As α is increased past 1, the false infeasible error rate decreases to less than 1% and the false feasible error rate steadily increases.

As with most classifiers, the accuracy of the BNC is heavily dependent on the number of training points. Figure 3.12 demonstrates how the BNC converges its accuracy with an increasing number of training points, for several values of α. After about 40 training points the BNC converges to an error rate below 10% for both the feasible and infeasible classifications, and decreases below 5% after 100 training points.

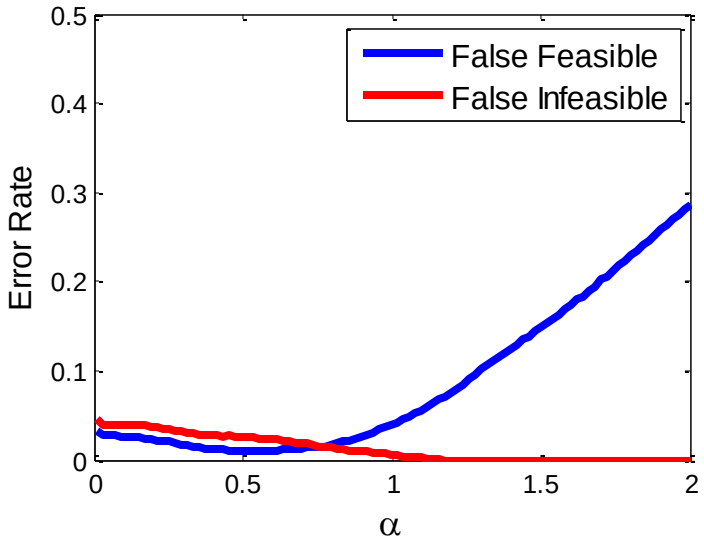

Figure 3.11. False feasible and infeasible classification error rate for a BNC with 50 training points.

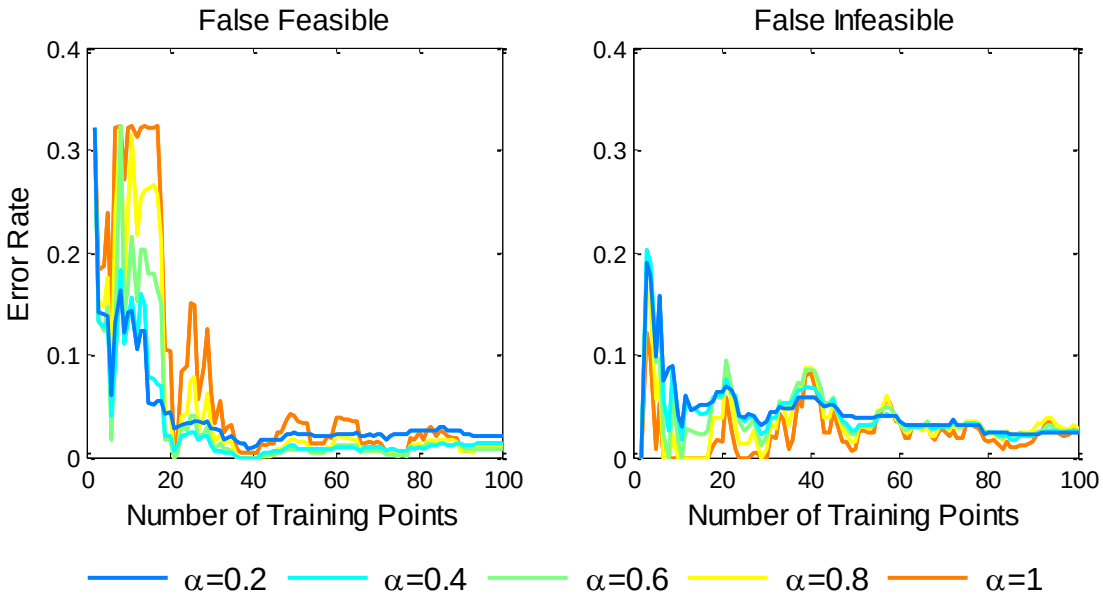

Figure 3.12. False feasible and infeasible classification error rate as a function of the number of training points.

The trend of the kernel bandwidth to classification accuracy from this spring design case study is that higher values of α result in lower false infeasible error rates and higher false feasible error rates. Although the error rate fluctuates in Figure 3.12 as the number of training points is increased, moderate values of α, such as in the range of 0.4-0.6, tend to yield more consistent and lower error rates. For this reason, the rule of thumb prescribed by Shahan and Seepersad of setting α to 0.4 is adopted. A supplementary strategy to improve the accuracy of the feasible decision boundary is to introduce a loss factor associated with each class, as defined by Equation 3.5, which scales the feasible and infeasible posterior probability distributions relative to each other.

3.4 Distinguishing the Attainable Performance Space

Although all of the points in the design space can be translated to points in the performance space, not all of the points in the performance space can be translated back to points in the design space. Therefore, in addition to classifying the feasible and infeasible performance spaces, the regions of the performance space that do not map to feasible or infeasible points within the design space must be classified as unattainable. This classification is particularly necessary to prevent designers from believing that they can attain higher performance values than they actually can. A second decision boundary for the performance space must be defined to separate the attainable and unattainable regions. That decision boundary is determined by first building a KDE on the attainable points using Equation 3.17, but since a KDE cannot be built on non-existent unattainable points, the boundary of the attainable space is set according to a probability threshold. A

point in the performance space is classified as attainable if the value of its KDE exceeds the probability threshold defined by Equation 3.22.

$$p_{threshold} = \prod_{i=1}^{m} \frac{1}{\hat{\sigma}_i \sqrt{2\pi}} e^{-\frac{\beta}{2}} \quad (3.22)$$

Where the variable β adjusts the influence of each known attainable point to the region around it. Increasing the value of β results in extending the attainable boundary outwards from a known attainable point, and thus increasing the percent of falsely classified attainable points in the performance space. Conversely, decreasing the value of β shrinks the attainable boundary inward, thereby increasing the percent of falsely classified unattainable performance points. The variable $\hat{\sigma}_i$ in Equation 3.22 represents the standard deviation of the training points in the i^{th} dimension. Due to the expectedly non-uniform spacing of training points in the performance space, β should be set based on the designer's preference in mapping either unknown, or well-studied regions of the space. Mapping unknown and sparsely populated regions of the performance space requires a high value for β, in order to extrapolate more from the training points. Conversely, in order to put less emphasis on the outlying training points and map well-studied regions, β should be set low because less extrapolation between points is needed.

Figure 3.13 depicts the posterior probability distributions of the feasible and infeasible classes, as well as the attainable probability threshold surface defined with β set to 2. Figure 3.14 shows the complete BNC classification of the performance space of both the feasible decision boundary and the attainable decision boundary. Decreasing β

will result in the probability threshold shown in Figure 3.13 to move vertically upwards and shrink the attainable space in Figure 3.14.

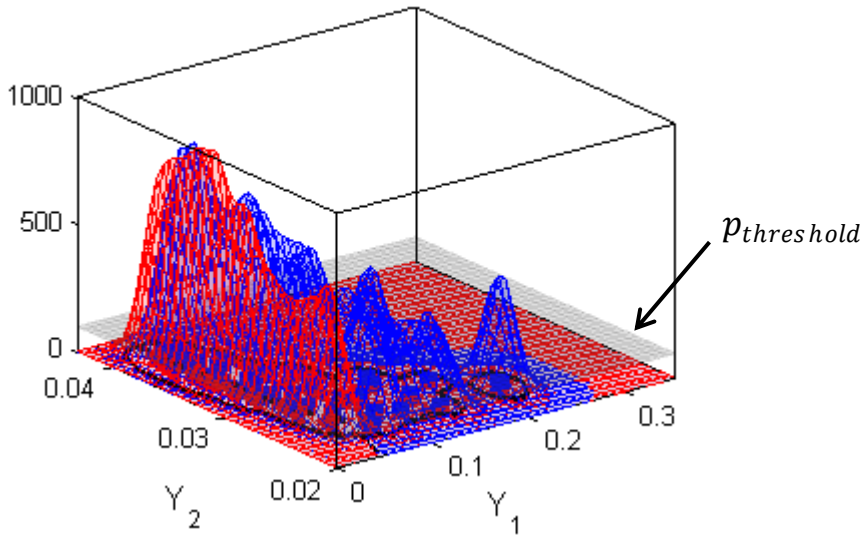

Figure 3.13. The posterior probability distributions for the feasible (blue) and infeasible (red) classes, in addition to the attainable probability threshold (grey).

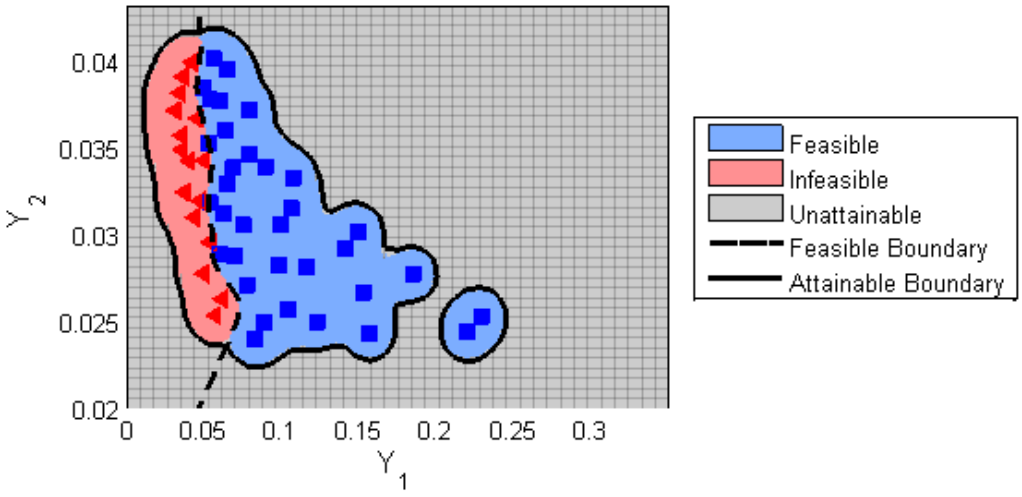

Figure 3.14. The BNC classification of the feasible and attainable boundaries.

To examine the accuracy of the classification of the attainable performance space, a set of 20,000 test points were drawn from a uniform distribution of the performance space. The test points were then compared to an approximated correct attainable boundary. The correct attainable boundary was approximated for this study by the set of outermost points in the performance space, based on a set of 100,000 samples drawn from a uniform distribution of the design space. For the accuracy study in this section, a point that has been classified by the BNC as attainable but is actually unattainable is defined to be a false attainable point. False unattainable points are defined to be classified by the BNC to be unattainable but are actually attainable. Figure 3.15 presents a summary of the false attainable and unattainable error rates as a function of the number of training points, for several values of β.

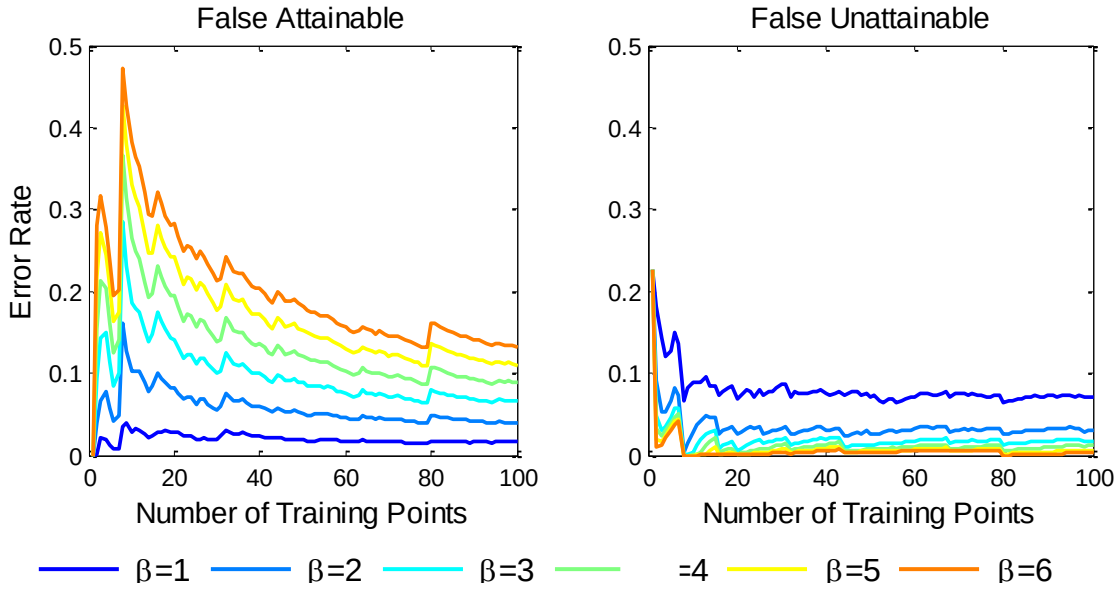

Figure 3.15. False attainable (left) and unattainable (right) classification error rates as a function of the number of training points.

The false attainable error rate decreases, as expected, as β decreases, which essentially shrinks the classified attainable space. The false unattainable error rate is low relative to the false attainable error rate, but experiences a rapid increase as β is decreased below 2 as a result of an overly conservative attainable boundary. From the false attainable error rate in Figure 3.15, it is inferred that an accurate attainable decision boundary requires more training points than defining the feasible/infeasible boundary, as the false attainable error rate continues to decrease after 100 training points. Setting the value for β will ultimately depend on the designer's preference on minimizing the false attainable vs. false unattainable errors.

3.5 Discussion

The BNC introduced in this chapter uses a set of training points to classify the performance space into feasible, infeasible, and unattainable regions. The classifier's tuning parameters of the kernel bandwidth, α, and the attainable probability threshold parameter, β, can be utilized to minimize the classifier's error. However, there is always going to be a tradeoff between minimizing the errors of each class. Additionally, the tuning parameters α and β do not act independently of each other. The interaction between α and β is examined in the study summarized in Figures 3.16 and 3.17, which uniformly sampled the performance space similar to the error rate calculation in Section 3.4 with a BNC having 50 training points, and defines the error rate of a particular class to be the ratio of the total number of misclassifications to the total number of samples. The false feasible and false infeasible error rates shown in Figure 3.16 show that at very

low values of β the error rates decrease to zero; however, this is due to the BNC classifying the entire space as unattainable, which causes the false unattainable error rate to increase as seen in Figure 3.17. A similar effect is seen for low values of α, which results in low false feasible error rates due to the peaky KDE that has probability values greater than the attainable threshold only at or very near to training points.

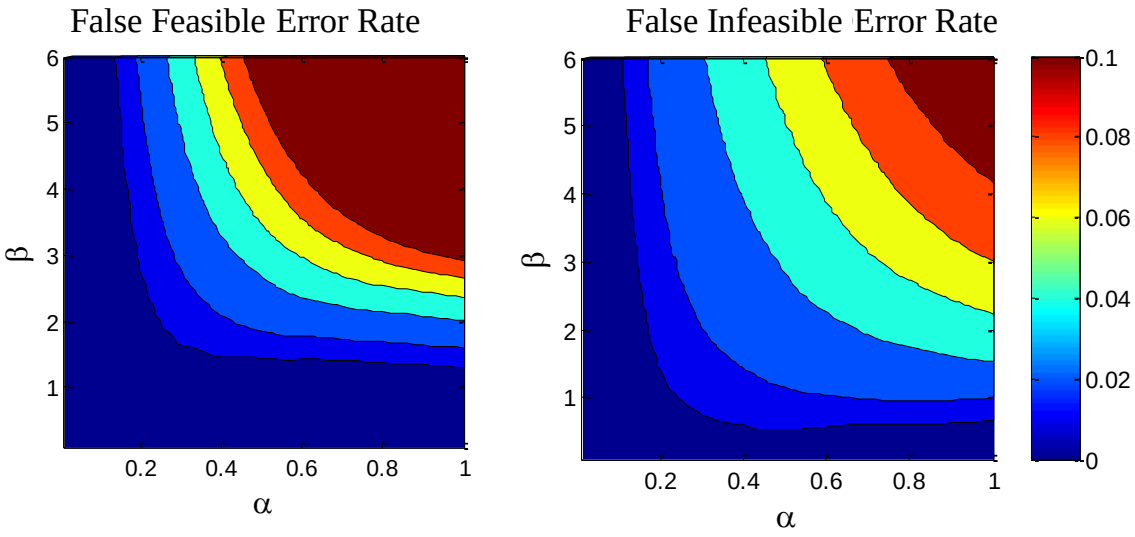

Figure 3.16. Total false feasible and infeasible classification error rates.

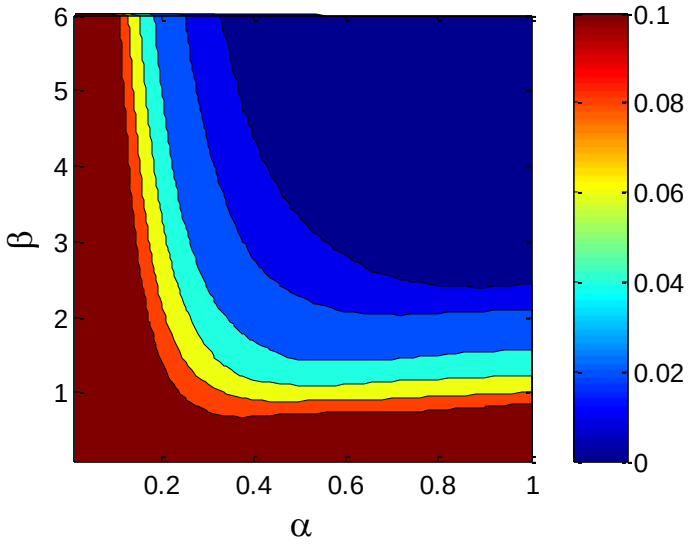

Figure 3.17. Total false unattainable classification error rate.

To maximize the usefulness of the proposed BNC, designers should define a cost of misclassification matrix, where the entry in row *i* and column *j* represents the cost of the BNC assigning class *i* when the actual class is *j* (Pearl, 1988). The cost of misclassification matrix can then be used as a metric for designers to consult when varying the tuning parameters α and β. For example, in the early stages of a design process it is desired to have a search space that includes all feasible designs even if some infeasible designs are included, too, in the interests of searching a broad design space. In this example, the cost of misclassifying a feasible design as infeasible should be set high while the cost of misclassifying an infeasible design as feasible should be set very low.

According to Shawe-Taylor and Cristianni (2004), an effective classification algorithm must exhibit three characteristics to be considered effective: (1) computational efficiency, (2) robustness to noise and error, and (3) statistical stability. In regards to the

computational efficiency of the method, to classify an unknown point using the BNC the probability distributions of both the feasible and infeasible classes must be computed. The computation of the probability distributions requires looping through all N training points, as well as an inner loop through all m dimensions of the performance space, according to Equation 3.17. This calculation yields a computational expense of order $\mathcal{O}(Nm)$, which may be negligible but can become computationally expensive as N and m increase together, commonly referred to as the "curse of dimensionality". The method effectively extracts pattern from noise, which can be seen by the decision boundaries capturing the irregularly shaped regions of the performance space with relatively low error rates. Additionally, the classifier has been seen to minimize the error as more training points are evaluated. The results shown by the proposed classifier meet the criteria defined by Shawe-Taylor and Cristianni, but more investigation is necessary to verify these claims under more intensive case studies.

Chapter 4: Quantifying Design and Performance Flexibility

A set-based design approach has been shown to be capable of adapting to changes in both design space constraints as well as performance criteria by maintaining *flexibility* until the late stages of the design process (Chang, Ward, Lee, & Jacox, 1994; Taguchi & Cariapa, 1993). Defining a set of designs that are feasible and satisfy a set of performance targets adheres to Sobek *et al*.'s (1999) set-based design guidelines, which then requires intelligent narrowing of the design space. In order to intelligently narrow the design space a method to accurately measure design and performance flexibility is needed. This chapter builds off of the techniques introduced in Chapter 3, which provided a framework to accurately represent and map a set of designs in both the design and performance space, to *quantify* the design and performance flexibility embodied in the set of designs.

Narrowing the design space can be a result of tightening performance criteria, resulting in a reduced set of designs that meet the tightened criteria. Tightening performance criteria prior to a thorough exploration of the design space may result in excluding high quality designs from ever being discovered. To intelligently narrow the design space, the performance criteria should be gradually tightened, so that resources can be focused on investigating the high performance region of the design space while only excluding the lowest quality designs. The design space may also be narrowed as a result of fixing one or more design variables. Fixing one or more design variables reduces the dimension of the design space, and allows for more efficient investigation of the design space, but should only be performed if it will not overly restrict the ability to find

satisfactory designs. A metric for design and performance flexibility must inform designers to the possible effects that narrowing the design space may have.

"Flexibility" has taken many definitions in design research, but in general is regarded as a sensitivity measure of the changes that can occur during the design process, while still maintaining the ability to find a satisfactory design (Ferguson, Siddiqi, Lewis, & de Weck, 2007). A metric for design flexibility should capture the degree to which a design can be changed while remaining feasible and satisfying a set of performance criteria. Following the work by Suh (1990), *design flexibility* is defined for this research as the proportion of the design space that produces satisfactory designs, relative to the size of the initial design space. Design flexibility, represented by $\mathbb{F}_D$, can be calculated according to Equation 4.1, where $\Phi(\Omega_X^{sat})$ represents the size of the satisfactory design space and $\Phi\left(\Omega_X^{init}\right)$ represents the size of the initial design space.

$$\mathbb{F}_D = \frac{\Phi(\Omega_X^{sat})}{\Phi\left(\Omega_X^{init}\right)} \quad (4.1)$$

Design flexibility is a measure of the amount that the design space has been narrowed, and is equal to 1 at the start of the design process and is equal to 0 once the design space has been narrowed to a single point. This definition of design flexibility is the inverse of the Information Content (IC) metric defined by Simpson *et al.* (1998), which quantified a designer's knowledge of the final design at a particular point in the design process. The two design spaces illustrated in Figure 4.1 exhibit two extremes of design flexibility, with the design space on the left of Figure 4.1 having high design

flexibility in both X_1 and X_2, and the design space on the right of Figure 4.1 having low overall design flexibility but high design flexibility in X_2.

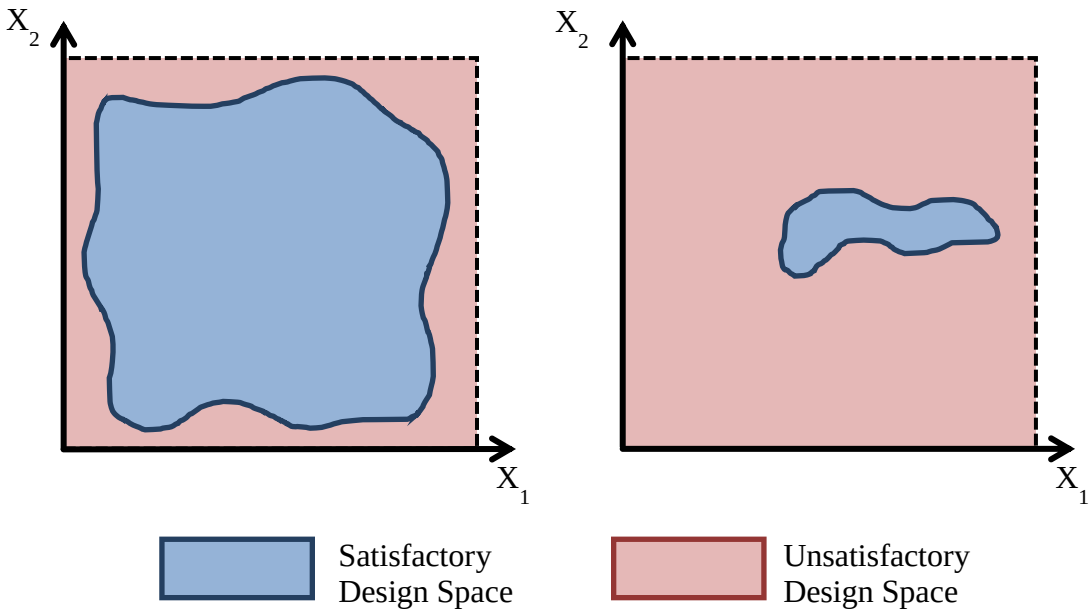

Figure 4.1 Illustration of a design space with high (left) and low (right) design flexibility.

In the performance space, a metric for flexibility should capture the ability of a set of designs to meet a ranged set of solutions, allowing for mitigation of unexpected changes in performance criteria (Chen & Yuan, 1998; Liu, Chen, Scott, & Qureshi, 2008). Sets of designs with high performance flexibility can meet a wide range of performance requirements, and thus easily adapt to changes in performance criteria. In this thesis, *performance flexibility* is defined as the proportional size of the feasible performance space associated with a set of designs that meets the set of performance requirements, relative to the size of the desired performance space.

The desired performance space is the space of performance variable values that designers want to achieve. Typically in a set-based design paradigm, designers seek to achieve a wide range of performance values in order to easily adapt to changes in performance requirements. The ability to meet a wide range of performance values also allows for enhanced product customizability. In some cases designers may simply define the desired performance space by assigning an upper and lower bound to each design variable. Designers can also move beyond the assumption of variable independence and take into account performance tradeoffs between variables and define an arbitrarily shaped desired performance space. In cases such as a multi-level design problem, the desired performance space may be defined by the feasible performance space of another design team, and have an irregular shape itself. Performance flexibility, represented as $\mathbb{F}_P$, is determined according to Equation 4.2, as a function of the size of the satisfactory performance space, $\Phi(\Omega_Y^{sat})$, and the size of the desired performance space, $\Phi\left(\Omega_Y^{des}\right)$.

$$\mathbb{F}_P = \frac{\Phi(\Omega_Y^{sat})}{\Phi\left(\Omega_Y^{des}\right)} \quad (4.2)$$

Performance flexibility gives designers an intuition of the consequences of a change in performance requirements. The performance space illustrated in the left of Figure 4.2 has high performance flexibility, because a large change in performance criteria, $Y_{1,target}$ and $Y_{2,target}$, can occur while still maintaining a set of designs that meet the new performance criteria. Conversely, the performance space illustrated in the right of Figure 4.2 has low performance flexibility, as a moderate change in performance criteria, $Y_{1,target}$, will result in an empty set of designs that meet the new set of

performance targets. Variable Y_2 shown in the performance space in the right of Figure 4.2 individually has high performance flexibility, because a wide range of desirable Y_2 values are achievable. Knowledge of performance flexibility with respect to each performance variable, and coupled relationships between performance variables, is helpful to designers in making decisions of where to focus their resources.

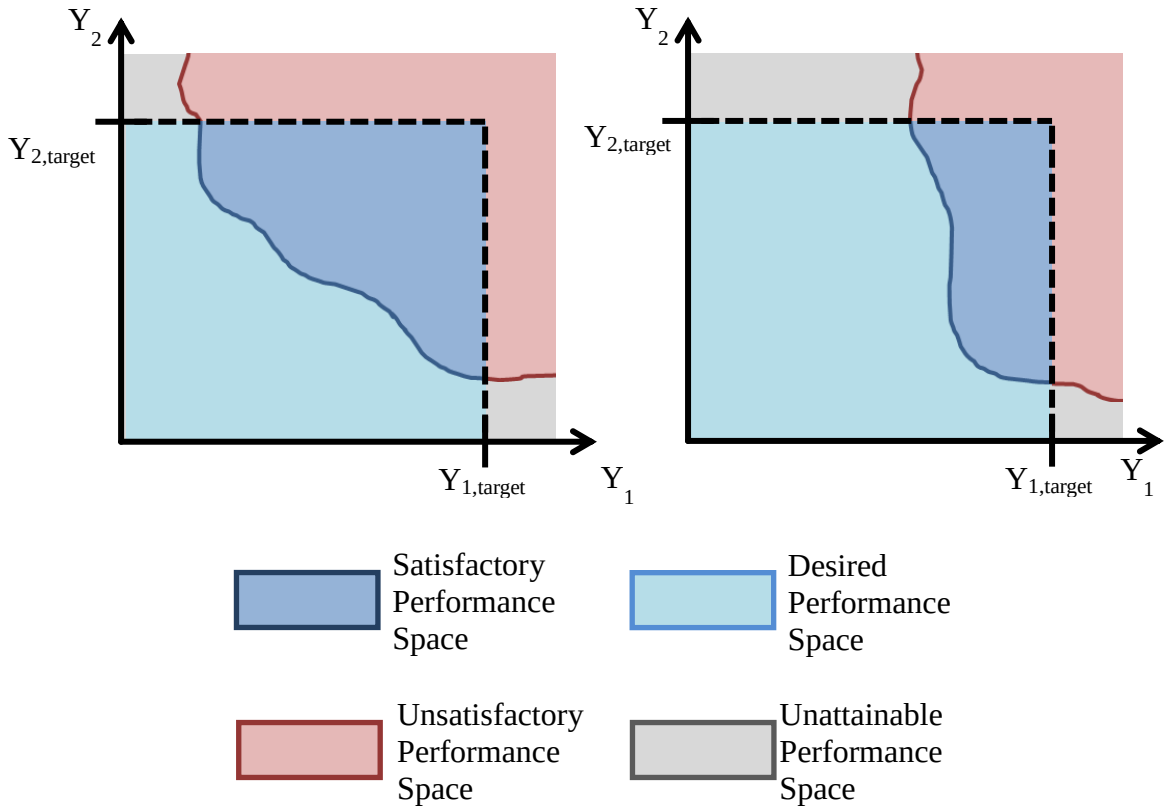

Figure 4.2. Illustration of a performance space with high performance flexibility in both Y_1 and Y_2 (left), and a performance space with low performance flexibility in Y_1 and high performance flexibility in Y_2 (right).

Classification accuracy has been found to be directly dependent on the method of set representation (Shahan & Seepersad, 2012). Consequently, the method of set representation directly impacts the accuracy of flexibility quantification. Section 4.1 presents the mathematical framework to quantify the size of a generic space. It will be seen that for the irregularly shaped spaces inherent to the satisfactory design and performance spaces, approximations will need to be made in order to follow the quantification framework in Section 4.1. Section 4.2 provides examples of two interval approximation techniques to quantify an irregularly shaped space, and highlights the limitations of interval methods of set representation. The Bayesian network classifier set representation method has been shown to have improved classification accuracy over the interval and probabilistic set representation methods, and will be used in a combination with a Monte Carlo integration method to quantify design and performance flexibility in Section 4.3. Following the method proposed in Section 4.3, a discussion on how to interpret and utilize the knowledge provided by an accurate quantification of design and performance flexibility is provided in Section 4.4.

4.1 Framework for Quantification

The "size" of a space is equivalent to the area of a space in two-dimensions, and volume in three-dimensions, and will be referred to as the hypervolume. The hypervolume, Φ, of a space, Ω,can be represented as a multivariate integral form according to Equation 4.3.

$$\Phi(\Omega) = \int_{\Omega} d\boldsymbol{x} \tag{4.3}$$

For an irregularly shaped and possibly discontinuous space, the bounds placed on the definite integral in Equation 4.3 are not likely to define a hyperrectangle. Instead, the hypervolume of Ω can be determined as a function of a space of known size, $\Re^m$, that contains the space of interest, Ω. The space of known size, $\Re^m$, can easily be created by assigning upper and lower bounds on each variable, thereby creating a hyperrectangular space. The hypervolume of the hyperrectangular space, $\Re^m$, can be determined by the definite integral in Equation 4.4, where the lower, x_i^{min},and upper, x_i^{max},bounds on each variable, x_i, define the integration bounds.

$$\Phi(\Re^m) = \int_{x_1^{min}}^{x_1^{max}} \int_{x_2^{min}}^{x_2^{max}} \dots \int_{x_m^{min}}^{x_m^{max}} dx_1 dx_2 \dots dx_m \tag{4.4}$$

The function that is being integrated in Equations 4.3 and 4.4 is a constant of 1 over the entire space, however, in order to determine the hypervolume of Ω as a subset of the hypervolume of $\Re^m$, an indicator function, $\varphi(\boldsymbol{x})$, is defined according to Equation 4.5.

$$\varphi(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in \Omega \\ 0, & \boldsymbol{x} \notin \Omega \end{cases} \tag{4.5}$$

The hypervolume of the irregularly shaped space can now be written as an integral over space $\Re^m$, according to Equation 4.6.

$$\Phi(\Omega) = \int_{\Re^m} \varphi(\boldsymbol{x}) d\boldsymbol{x} \quad (4.6)$$

The challenge in integrating Equation 4.6 is that the indicator function typically cannot be written in a closed form. The following methods presented in this Chapter make approximations for integrating Equation 4.6. It will be seen that an effective method to quantify design and performance flexibility must have an accurate approximation for integrating Equation 4.6 in a computationally efficient way.

4.2 The Need for an Efficient High Fidelity Quantification **Method**

There are slight differences in the methods to quantify design flexibility and performance flexibility. In both cases, the flexibility metrics are framed as ratios of the size of the satisfactory space to the size of the "whole" space – the primary difference between the metrics being the definition of the "whole" space. In the design space, the "whole" space is the initial design space. Often the initial design space is a hyperrectangular space defined by placing minimum and maximum bounds on each design variable. However, in cases such as a multi-level design process, the initial design space may be specified as the feasible performance space from a lower-level design team, and may take an irregular shape. In the performance space, the "whole" space is the desired performance space corresponding to the range of performance values that meet the performance requirements. The desired performance space may have bounds placed on it from another design team, but if not, finite bounds must be assigned by the designer.

This section will demonstrate the limitations of interval and grid set representation methods in quantifying the size of an irregularly shaped space using a helical spring design problem, which is defined in Appendix A. In this example spring problem the design variables X_1 and X_2 correspond to the spring's coil diameter and wire diameter, respectively. The performance variables for the spring design problem, Y_1 and Y_2, represent the spring compression under a target loading and the total spring width, respectively. For this design problem both performance variables, Y_1 and Y_2, are attempted to be minimized, with satisfactory designs having performance values lesser than the performance targets, $Y_{1,target}$ and $Y_{2,target}$. Additionally, constraints g_1 and g_2 correspond to the maximum allowable shear stress and minimum allowable spring index, respectively, which are used to define a design's feasibility.

A BNC of the design and performance spaces is created following the methods described in Chapter 3 of this thesis, with tuning parameters α and β set to 0.3 and 2, respectively. The BNC was trained by 100 sample points drawn from a uniform distribution in the design space according to the Halton sequence (Freeman & Halton, 1951). The bounds to the design and performance space are given in Table 4.1, which are then used to normalize each design and performance variable to range from 0 to 1.

Table 4.1. Design and performance variable bounds.

	X_1 (m)	X_2 (m)	Y_1 (m)	Y_2 (m)
Minimum	0.0178	0.0033	0.0000	0.0200
Maximum	0.0033	0.0048	0.3500	0.0430

Normalization of the design and performance spaces allows for each variable to be equally weighted in the quantification of the space. Normalization is especially crucial when variables have widely varying units and ranges. Assigning bounds to the performance space must be somewhat arbitrarily made because the attainable range of each performance variable is typically not known at the start of the design process. An approximation of the bounds of the performance space is adequate, and can always be updated as more knowledge of the space is gained.

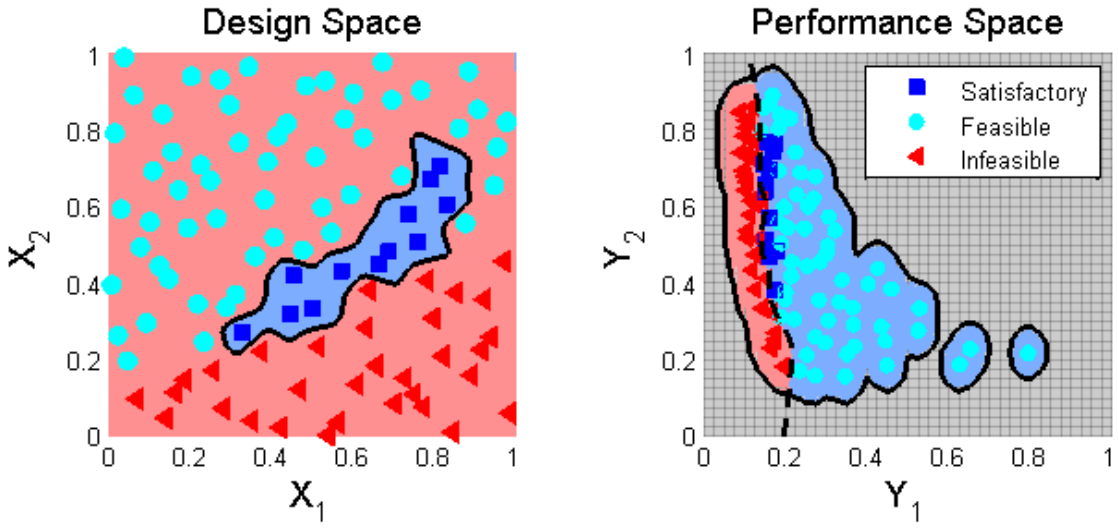

Figure 4.3. The BNC mapping of the design space (left) and performance space (right) of the spring design problem with 100 training points.

The design and performance space mappings for the spring design problem are shown in Figure 4.3, with the blue square points corresponding to the feasible designs that also meet the performance requirements, the cyan circle points corresponding to feasible designs that do not meet the performance requirements, and the red triangle

points corresponding to the infeasible points that violated one or more constraints. In the design space shown in Figure 4.3, the light blue background represents the satisfactory design space and the light red background represents the unsatisfactory design space, as defined by the BNC mapping. In the performance space in Figure 4.3, the light blue background represents the feasible performance space, the light red background represents the infeasible performance space, and the grey background represents the unattainable performance space, as defined by the BNC. From the feasible performance space the satisfactory performance space can be trivially identified, as the values of performance variables, Y_1 and Y_2, that are less than specified performance requirements, which are defined *in* the performance space.

The most common and elementary method for quantification of the design space is through intervals. Intervals are defined by assigning a range for each variable independently. Intervals are often used in defining a hyperrectangular initial design space, but suffer errors when used to quantify an irregularly shaped space. For instance, an interval of the satisfactory design space in the helical spring problem, shown in Figure 4.3, is typically defined by the minimum and maximum parameter values for known satisfactory designs.

The volume of the satisfactory design space, $\Phi(\Omega_X^{sat})$, can be approximated for an m-dimensional design space using an interval according to Equation 4.7. Letting $\boldsymbol{X}^{sat}$ represent the set of satisfactory designs, with each design instance being composed of an m-dimensional array, $\boldsymbol{x} = [x_1, x_2, \dots, x_m]$, the variable $\boldsymbol{x}_i^{sat}$ in Equation 4.7 represents an array corresponding to i^{th} variable values from the set of satisfactory designs.

$$\bar{\Phi}(\Omega_X^{sat}) = \int_{x_{1,min}^{sat}}^{x_{1,max}^{sat}} \int_{x_{2,min}^{sat}}^{x_{2,max}^{sat}} \dots \int_{x_{m,min}^{sat}}^{x_{m,max}^{sat}} dx_1 dx_2 \dots dx_m = \prod_{i=1}^{m} \left(x_{i,max}^{sat} - x_{i,min}^{sat}\right) \quad (4.7)$$

The satisfactory design space of the helical spring problem is shown in Figure 4.4 with an interval representation drawn according to Equation 4.7 as well as the boundary defined by the BNC as shown in Figure 4.3. The interval in Figure 4.4 defines a satisfactory design space that is much larger than the actual satisfactory design space, as the falsely classified satisfactory design space is nearly as large as the correctly classified satisfactory design space. A more conservative interval could be used to define this interval, but would then increase the size of the falsely classified unsatisfactory design space.

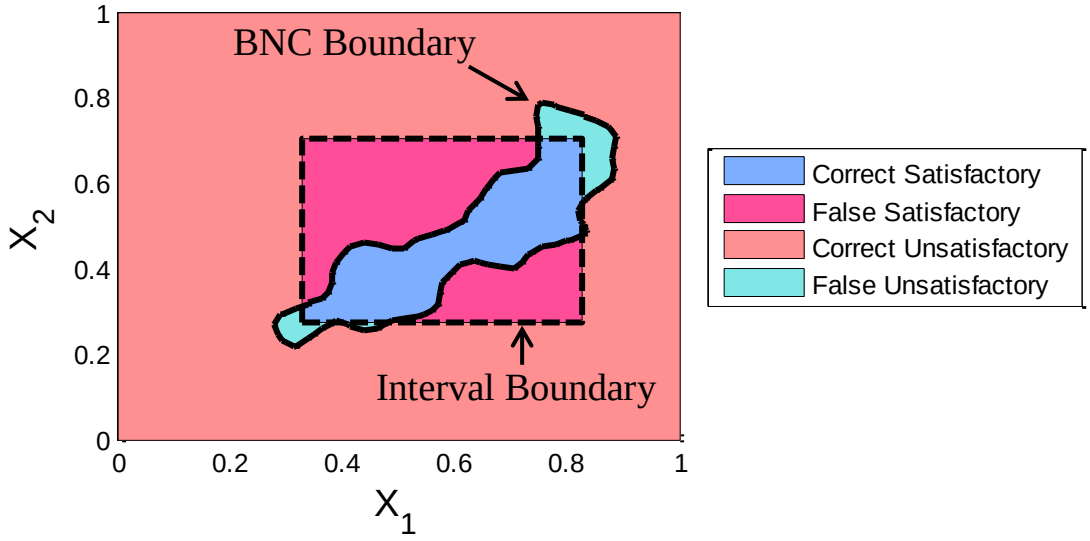

Figure 4.4. Interval quantification of the satisfactory design space.

The accuracy of the interval representation can be improved by dividing the space into an m-dimensional hyperrectangular grid. Each grid segment can be classified according to various criteria, such as whether or not a satisfactory training point exists within that sub-interval, whether there are more satisfactory training points than unsatisfactory training points within that sub-interval, etc. For the purpose of this demonstration, a sample point is generated at the geometric center of each grid segment, and is classified using the BNC. The sub-interval is then classified according to the class of the sample point returned by the BNC, which provides a more accurate interpretation of the space than the set of training points alone (Shahan & Seepersad, 2012).

To use the grid method to quantify the size of the satisfactory design space, each of the m design variables is first divided into g segments. The overall space is then divided into g^m hyperrectangular spaces, requiring $n = g^m$ sample points. If the design space has been normalized such that the total volume is equal to 1, then the volume of each grid segment is equal to $1/g^m$. The volume of the satisfactory design space can then be approximated as the number of sample points that were classified as satisfactory, multiplied by the volume of an individual grid segment, according to Equation 4.8.

$$\bar{\Phi}(\Omega_X^{sat}) = S\left(\frac{1}{g^m}\right) = \frac{S}{n} \quad (4.8)$$

Where the variable, S, represents the number of sample points that were classified as satisfactory. Using the indicator function defined in Equation 4.5, which is equal to 1 if the sample point is classified as satisfactory and equal to 0 otherwise, the value of S can be calculated according to Equation 4.9.

$$S = \sum_{i=1}^{n} \varphi(\boldsymbol{x}^i) \qquad (4.9)$$

The satisfactory design space of the helical spring problem is quantified using the grid method in Figure 4.5. The grid method is seen to have a reduction in false unsatisfactory classification relative to the interval method. Additionally, by comparison of the quantification with 5 divisions per variable and 10 divisions per variable, it is clear that the error can be decreased further by increasing the number of divisions per variable, g. However, a major drawback of the deterministic grid sampling method is that once the number of divisions has been set, it is difficult to change this without creating a completely new grid and resampling the new points.

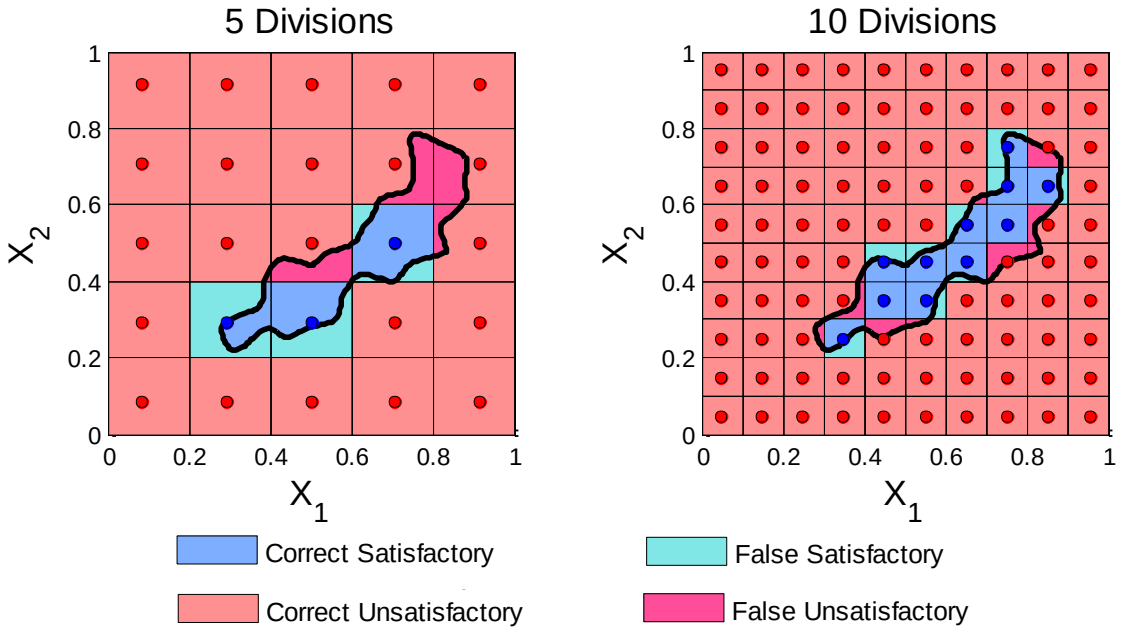

Figure 4.5. Grid quantification of the satisfactory design space with 5(left) and 10 (right) divisions per variable.

The upper bound of the absolute error, ε, for the volume approximation in Equation 4.8 can be defined as a function of the boundary surface area of the space, $\rho(\Omega_X^{sat})$, and the number of divisions along each variable, g, according to Equation 4.10 (Fishman, 1996). Although the boundary surface area, $\rho(\Omega_X^{sat})$, will be likely unknown, a generous approximation for it will allow Equation 4.9 to be approximated.

$$\varepsilon = |\bar{\Phi}(\Omega_X^{sat}) - \Phi(\Omega_X^{sat})| \leq \frac{\rho(\Omega_X^{sat})}{g} = \frac{\rho(\Omega_X^{sat})}{n^{1/m}} \quad (4.10)$$

Therefore, in order to guarantee an absolute error no larger than ε, the total number of sample points required can be determined by Equation 4.11. The operator $\lceil\theta\rceil$ in Equation 4.11 rounds up the value of θ to the next integer, and is necessary to ensure that Equation 4.10 returns a non-fractional number of samples.

$$n(\varepsilon) = \left\lceil \left(\frac{\rho(\Omega_X^{sat})}{\varepsilon} \right)^m \right\rceil \quad (4.11)$$

To analyze the efficiency of this method, suppose that the dimensionality of the space, m, is increased while maintaining a constant surface area of the space. In order to maintain the same maximum absolute error, it can be inferred from Equation 4.11 that the number of divisions, g, of each variable remains constant. However, the number of samples required grows exponentially by order $\mathcal{O}(\varepsilon^{-m})$. This exponential growth in samples required highlights the limitation of this quantification method for moderate and large number of variables, m. Additionally, in this thought experiment the boundary surface area of the space was held constant, which is a very conservative estimate. If the

boundary surface area increases with m, the number of samples required grows even faster.

The computational limitation to the grid approach for representing and quantifying an irregularly shaped space further motivates the use of the BNC to map the space. Additionally, without using the BNC to classify the sample points in the grid quantification method, there is an inherent maximum number of divisions per variable. If the grid sample points are classified based on whether or not a satisfactory training point lies within a sample's grid space, the number of divisions per axis will be limited by requiring a training point to fall within each grid space. The benefit of the BNC is clearly seen here as it provides an accurate interpolation of knowledge to unexplored regions of the design space. Although the BNC could be used in conjunction with the grid quantification method to achieve high quantification accuracy, the following section presents a significantly more computationally efficient method to do this.

4.3 A Monte Carlo Method for Quantifying Design and Performance Flexibility Using Bayesian Network Classifiers

The Monte Carlo (MC) method provides a framework to approximate the hypervolume integral in Equation 4.3 through statistical sampling. The MC method to approximating the integral in Equation 4.3 is similar to the grid approximation method described in Section 4.2, with the deterministic sample points in the grid approximation method replaced by a sequence of stochastic samples. The sample points are then classified using the BNC. The MC error in approximating the integral in Equation 4.3

decreases as the number of samples, n, is increased at the rate of $n^{-1/2}$, and is independent of the dimensionality of the problem (Evans & Swartz, 2000; Hammersley, 1960). This property of the MC method presents a substantial efficiency improvement over the grid method, which decreases in error roughly at the rate of $n^{-1/m}$, especially as the number of variables, m, becomes large.

To approximate the hypervolume of a generic space, Ω, as a subset of a normalized m-dimensional space, $\Re^m$, using the MC method, a sequence of independent random samples must be drawn from a uniform probability distribution. Each sample is evaluated and classified with the BNC, which is used to define the value of the indicator function, $\varphi(\boldsymbol{x}^i)$, according to Equation 4.5. As a consequence of the sequence of samples being random and independent, the indicator function, $\varphi(\boldsymbol{x})$, in Equation 4.12 is an independent Bernoulli random variable. The indicator function therefore has the properties given in Equations 4.12 and 4.13, regarding the probability of the value that the indicator function will take for a random sample $\boldsymbol{x}^j$.

$$p(\varphi(\boldsymbol{x}^j) = 1) = \int_{\Re^m} \varphi(\boldsymbol{x}) d\boldsymbol{x} = \Phi(\Omega) \quad (4.12)$$

$$p(\varphi(\boldsymbol{x}^j) = 0) = \int_{\Re^m} d\boldsymbol{x} - \int_{\Re^m} \varphi(\boldsymbol{x}) d\boldsymbol{x} = 1 - \Phi(\Omega) \quad (4.13)$$

The sum of the indicator function values, S, across the set of n samples is defined in the same manner as it was in the grid quantification method, according to Equation 4.9. By definition, the sum of a sequence of Bernoulli random variables, S, follows a binomial

distribution. As a property of a binomial distribution, the probability that S will equal a number i, after n samples is defined by Equation 4.14.

$$p(S = i) = \binom{n}{i} \left(\Phi(\Omega)\right)^{i} \left(1 - \Phi(\Omega)\right)^{n-i} \quad (4.14)$$

Additionally, the expected value of the binomial distribution, S, is defined according to Equations 4.15.

$$E[S] = n\left(p\left(\varphi\left(\boldsymbol{x}^{j}\right) = 1\right)\right) = n\Phi(\Omega) \quad (4.15)$$

From the expected value of S in Equation 4.15, the hypervolume of Ω can be approximated after n samples according to Equation 4.16.

$$\bar{\Phi}(\Omega) = \frac{1}{n}\sum_{i=1}^{n} \varphi\left(\boldsymbol{x}^{i}\right) = \frac{S}{n} \quad (4.16)$$

According to the law of large numbers, as n is increased to infinity the approximation of the hypervolume, $\bar{\Phi}(\Omega)$, in Equation 4.16 converges to the actual hypervolume, $\Phi(\Omega)$ (Fishman, 1996). Therefore, $\bar{\Phi}(\Omega)$ is an unbiased, strongly consistent estimator of $\Phi(\Omega)$.

The *standard error* of the approximated hypervolume, $\bar{\Phi}(\Omega)$, serves as a rough estimate of the statistical error associated with this approximation. The standard error is defined as the square root of the approximated variance in the approximated hypervolume, according to Equation 4.17 (Fishman, 1996).

$$\sqrt{Var\left(\bar{\Phi}(\Omega)\right)} = \sqrt{\frac{\left(\frac{S}{n}\right)\left(1-\frac{S}{n}\right)}{n-1}} \quad (4.17)$$

The standard error is essentially a measure of how much the approximated hypervolume changes with each subsequent sample. Although there is no definitive rule correlating the standard error to the true error, it can be inferred that a lower value of the standard error requires less samples before $\bar{\Phi}(\Omega)$ becomes a good approximation of $\Phi(\Omega)$. The ability to approximate the error in the approximated hypervolume of Ω is a very useful property in order to know when to stop taking samples. There are additional metrics that are not discussed in this thesis, but can be found in (Fishman, 1996), that give error bounds to $\bar{\Phi}(\Omega)$ as well as give a worst case estimate of the number of samples necessary to have an absolute error less than a given value, ε.

The MC method is applied to approximating the design flexibility of the helical spring design problem according to Equation 4.16. Figure 4.6 shows the design space after 100 samples (left) and 500 samples (right), with the indicator function $\varphi(\boldsymbol{x})$ coloring the sample points as blue if they are inside the satisfactory design space and red otherwise. The sample points were drawn from the pseudorandom Halton sequence. To prevent any of the sample points to correspond to any of the training points, which were also drawn from the Halton sequence, the first 1,000 points in the Halton sequence were discarded for the MC samples. The use of the Halton sequence to generate the MC sample points has the benefit over a purely random sequence in that it is a space filling algorithm, and does not cluster points together. This modification to the basic MC algorithm increases the efficiency of the number of sample point evaluations required to

converge to an accurate approximation of the integral in Equation 4.6. The Matlab® code used to implement this method is included in Appendix B.

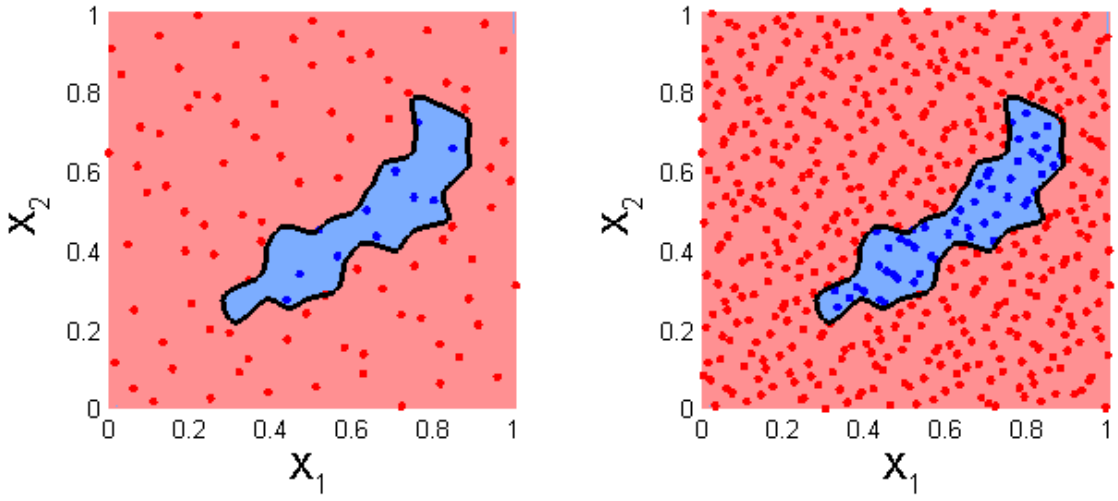

Figure 4.6. Monte Carlo samples of satisfactory design space 100 samples (left) and 500 samples (right).

The MC method presented to approximate the size of the satisfactory design and performance spaces samples the BNC mapping rather than performing new concept evaluations. Sampling the BNC requires minimal computational expense compared to the concept evaluation process, but limits the accuracy of the MC approximation to the accuracy of the BNC mapping. To isolate and analyze the error of the MC algorithm alone, the "true" quantification value is taken to be the value returned by the MC algorithm after 50,000 sample points. The true error is therefore taken to be the difference between the size of the space approximated by the MC algorithm to the size of the space defined by the BNC mapping.

The design flexibility of the helical spring problem was found to be 0.104. The MC approximation of the design flexibility is shown in Figure 4.7 to be fairly accurate even after 10 samples, and converges to better than 99% accuracy after 10,000 samples. The true error plot in the Figure 4.7 has frequent dips in error, which occurs when the samples happen to very accurately predict the design flexibility. The dips in error, however, are random in nature, and therefore the peaks in the error plot are a more accurate depiction of the true error. The predicted reduction in error as a function of training points, $n^{-1/2}$, is plotted in the dashed black line in the right plot of Figure 4.7, and very accurately approximates the true error. The standard error, plotted in blue, approximates the true error very closely too, but is an underestimation.

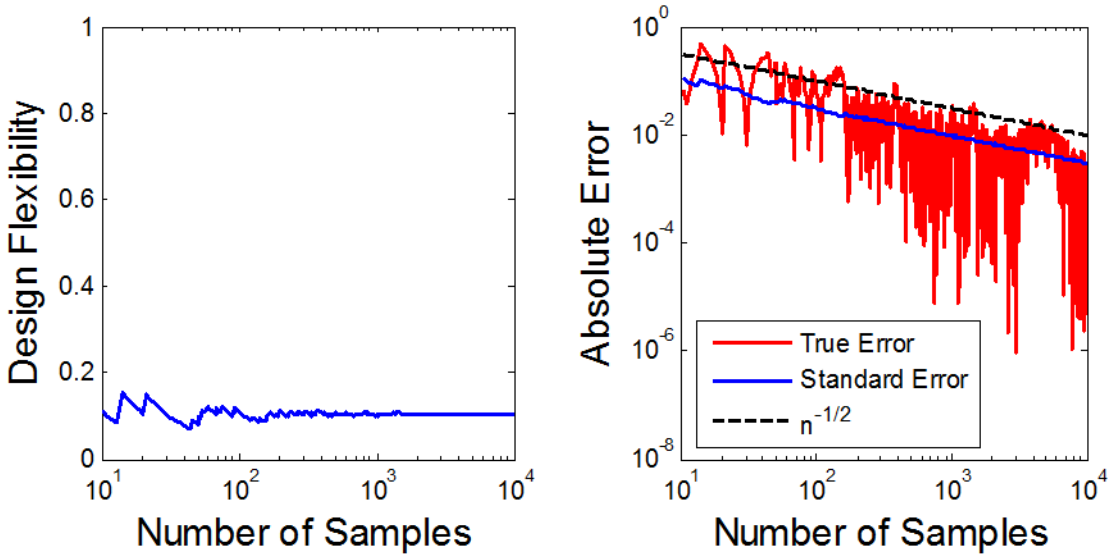

Figure 4.7. Monte Carlo approximation of the design flexibility (left) and error (right) as a function of the number of samples.

The MC approximation of the performance flexibility of the helical spring problem is demonstrated in Figure 4.8. The desired performance space is shown highlighted in light blue in the left plot in Figure 4.8, and normalized in the right plot in Figure 4.8. The MC samples were drawn from the same Halton sequence as in the design space. The desired performance space shown in the right plot of Figure 4.8 illustrates how the indicator function only captures the satisfactory performance space, and lumps the infeasible and unattainable performance spaces together as shown in red.

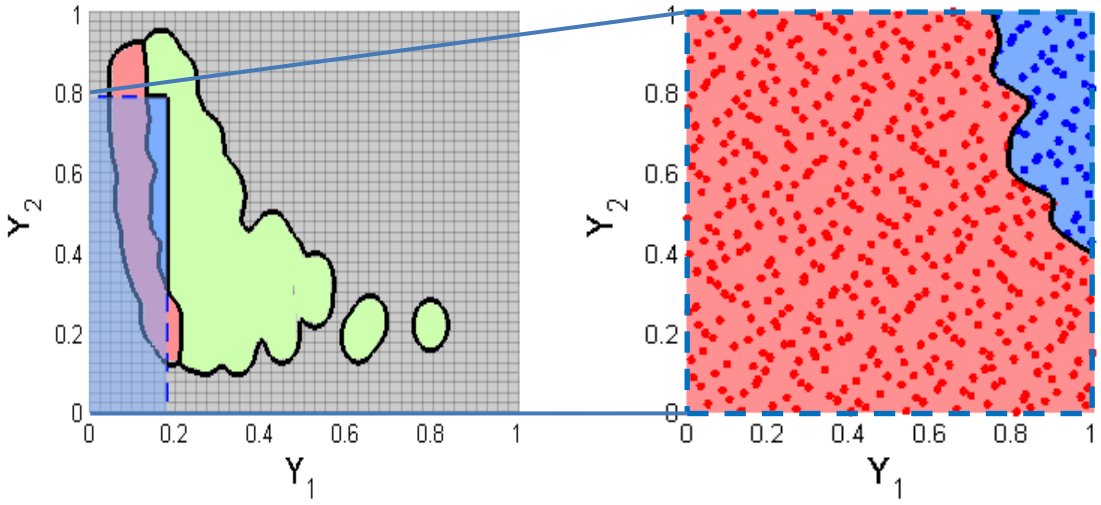

Figure 4.8. Monte Carlo samples of attainable performance space (left) and satisfactory performance space (right), each with 500 sample points.

The performance flexibility of the helical spring problem was found to converge to roughly 0.107 after 10,000 samples, as shown in Figure 4.9. The standard error is seen to again underestimate the true error in the performance flexibility, with the error prediction of $n^{-1/2}$, proving to be an excellent estimation of the true error. Although both

the design flexibility and performance flexibility were found to be very similar in this helical spring problem, this relationship is coincidental. The performance flexibility is measured as a percentage of the size of the satisfactory performance space to the desired performance space, in which the lower bounds to the desired performance space were somewhat arbitrarily approximated, and could be easily changed.

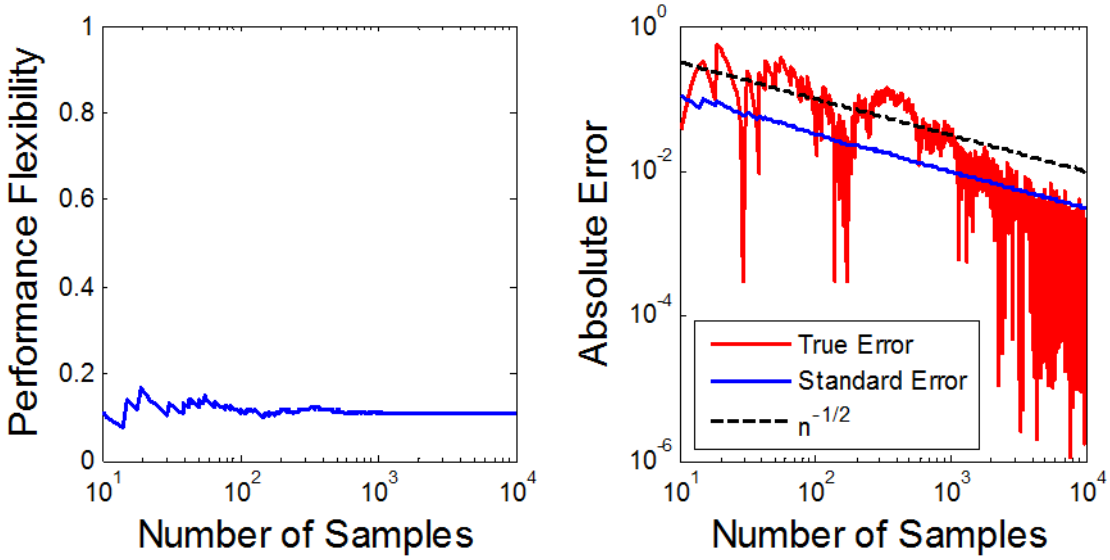

Figure 4.9. Estimate error of Monte Carlo approximation with standard error.

The MC method to quantify design and performance flexibility using a BNC mapping of the space is both accurate and computationally efficient. While the error in the flexibility quantification method can come from both the BNC mapping and the MC algorithm, with enough sample points for the MC algorithm the error can be isolated to the BNC mapping alone. The use of the Halton sequence to generate the stochastic sample points for the MC algorithm increases the efficiency of the algorithm by reducing the variance of the MC integral approximation. Many other variance reduction methods

exist to intelligently choose samples, thereby requiring less samples to attain a comparable accuracy (Skowronski & Turner, 1997). The MC method can also give designers real time feedback on the accuracy of the design/performance flexibility, allowing for custom bounds to be placed on the maximum error.

4.4 Discussion

The MC method of sampling the BNC mapping of the design and performance spaces gives designers an ability to leverage design and performance flexibility in intelligently narrowing the design space. As the design space is narrowed, the design and performance flexibility metrics can be used to minimize any adverse effects. Ideally the decision to narrow the design space can be framed as a multi-objective optimization problem, in which the designer seeks to reduce design flexibility while maximizing performance flexibility. One way to do this is to perform a parameter sweep across the performance requirement thresholds, while keeping track of the resulting design and performance flexibility values. Figure 4.10 shows a zoomed in view of the desired performance space, with design and performance flexibility contours drawn equal to 75%, 50%, and 25% of the original design and performance flexibility. The reduction in design flexibility in each of the contours in Figure 4.10 results from tightening the performance requirements, and having fewer training points being classified as satisfactory. The differences in the contours of design and performance flexibility in Figure 4.10 are due to the difference in distribution of training points in the design and

performance spaces. These differences can be exploited to narrow the design space in an intelligent manner.

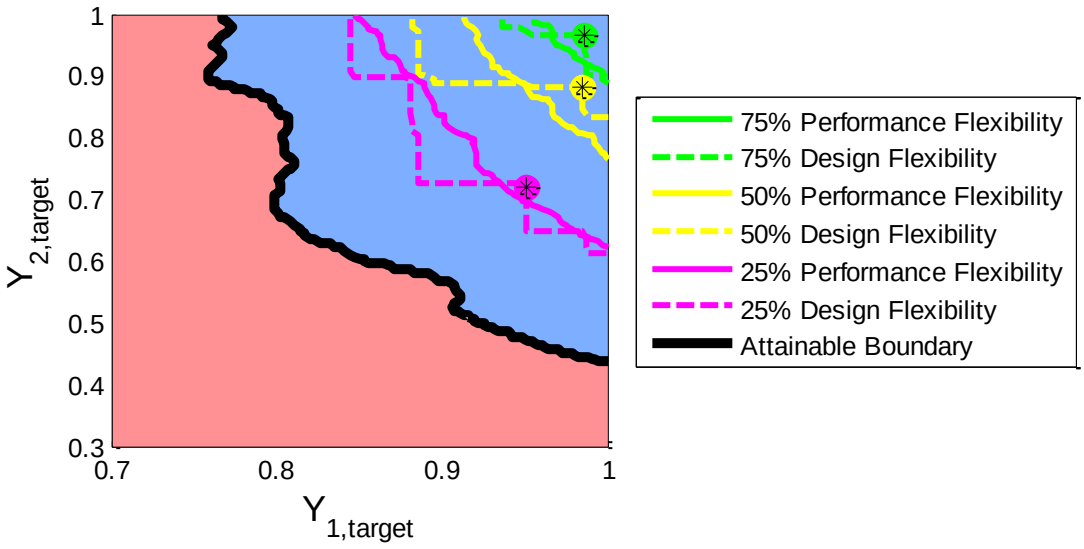

Figure 4.10. Contours of constant design and performance flexibility, shown in a normalized, zoomed-in desired performance space.

Adhering to the goal of maximizing performance flexibility when narrowing the design space, the circles with an asterisk plotted in Figure 4.10 represent the largest value of performance flexibility on the contour of constant design flexibility. Therefore, the points identified by the circle and asterisk are the points the values that a designer should assign to the performance requirements in order to gradually narrow the design space. The performance flexibility values at these points, corresponding to design flexibility values of 0.75, 0.50, and 0.25, are 0.92, 0.75, and 0.30, respectively. This demonstrates a considerable benefit in performance flexibility when narrowing the design space, if the performance requirements are tightened in this manner. The corresponding design and

performance spaces for this example of tightening performance requirements are shown in Figure 4.11. This example of intelligently narrowing the design space would not have been possible without an effective flexibility quantification method.

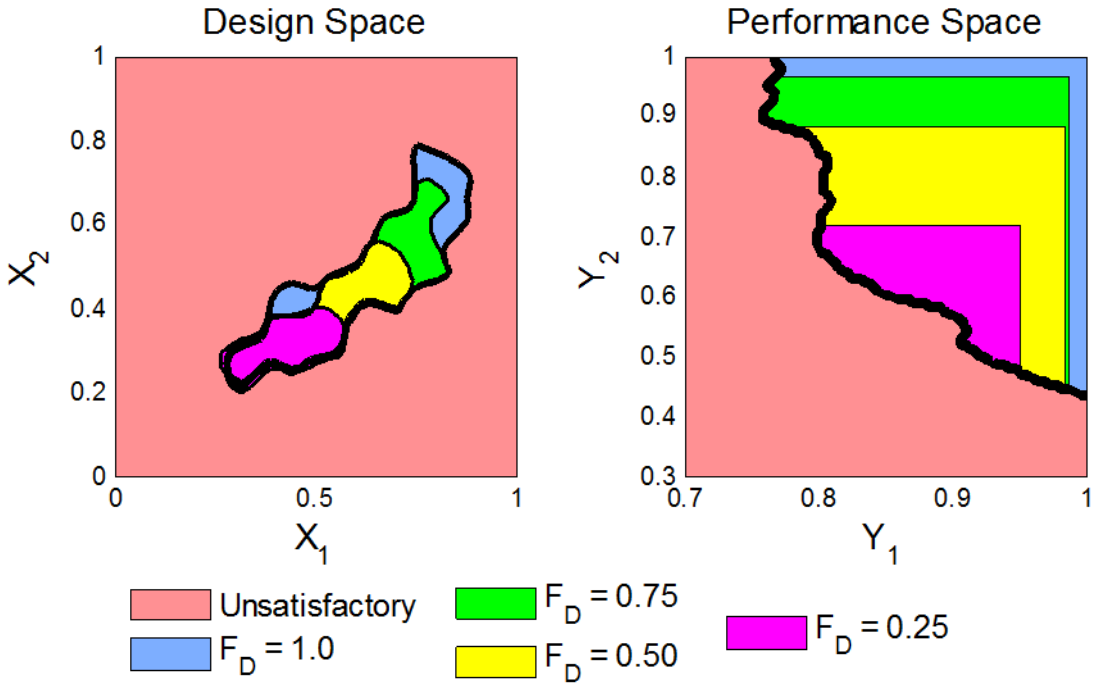

Figure 4.11. Updated design and performance spaces resulting from a change in performance requirements.

Design and performance flexibility can also be used as a sensitivity measure by artificially changing performance requirements to observe the impact of a single or coupled change in one or more performance criteria. The performance flexibility during a sensitivity analysis can give insights as to the most vulnerable performance criteria to unexpected changes. Similarly, design flexibility gives insight into how much the satisfactory design space may be narrowed as a result of a change in performance criteria.

In a design process, a designer cannot be satisfied with a set of designs that have very low performance flexibility. Consumer demands can change both unexpectedly and drastically. A change in performance criteria that reduces performance flexibility to zero will likely cause a timely setback to the design process, forcing the designer to expand the design space. It may be cost effective to spend extra resources in exploring regions of the design space that will increase performance flexibility, and prevent potential setbacks in the later stages of the design process. The key is to plan ahead, prepare for untimely and undesired events, and do not wait until it is too late to leverage flexibility.

Chapter 5: Hierarchical Materials Modeling and Design Study

The applications of the Bayesian network classifier (BNC) proposed in this thesis have thus far been limited to mapping the performance space of a single-level problem. In a single-level design problem, it is sufficient to map the regions of the *design* space that offer satisfactory performance values. In a multi-level design problem, however, where the performance space of an upstream design team corresponds to the design space of a downstream design team, it becomes necessary to create a mapping of the satisfactory performance space of the upstream design team. As part of a set-based design strategy (Sobek, Ward, & Liker, 1999), BNCs are coupled with an exploratory search to identify and map regions of the design space with desirable characteristics at each hierarchical level. Those maps are then intersected across hierarchical levels to identify satisfactory system-wide designs. As opposed to bottom-up, deductive hierarchical *modeling*, the BNC approach is a top-down, inductive *design* approach that is driven by meeting high-level system performance requirements.

This chapter demonstrates the utility of the BNC performance space mapping through a multi-level material design problem. The BNC approach yields significant insights about the behavior of the material system at each level, and supports rapid identification and exploration of promising multi-scale solutions. The following section presents the technical background for the composite material design problem and outlines the modeling that will be conducted at each hierarchical level.

5.1 Background of Hierarchical Material Modeling

Composites consisting of negative stiffness particulate heterogeneities embedded in a continuous viscoelastic host material have been theoretically (Lakes, 2001) and experimentally (Lakes, Lee, Bersie, & Wang, 2001) shown to display drastic enhancements in effective damping with the potential to maintain or even increase the overall stiffness of the host material. These types of materials are of significant interest to the engineering community because of their potential for improving the vibro-acoustic performance of structures in aerospace, automotive, and marine industries (Haberman, Berthelot, & Cherkaoui, 2006; Koutsawa, Haberman, Daya, & Cherkaoui, 2008).

Modeling and design of these material systems must occur on three distinct scales: the micro-, meso-, and macroscales, as illustrated in Figures 5.1 and 5.2. The microscale is defined by the smallest-scale geometry of interest: the internal structure of the negative stiffness inclusions. The microscale inclusions exploit the unique non-monotonic force-displacement nature of bistable systems to produce highly absorptive composite materials. Such bistable systems are exemplified by an axially compressed beam, which, under transverse loading, initially exhibits positive stiffness until it buckles and consequently "snaps-through" to its second stable configuration. During this transition, the beam exhibits negative stiffness when loaded with displacement control. Taking inspiration from a buckled beam, the inclusion design presented here mimics such a system on four of its six cubic faces to induce negative stiffness in two orthogonal directions. This negative stiffness behavior leads to high levels of energy absorption and mechanical loss when the composite material is subjected to vibro-acoustic loading. The

microscale inclusions presented in this paper are unique in that they employ a thermal expansion mismatch to induce negative stiffness behavior. More details of the geometry and fabrication of the inclusions are included in Section 5.2.1 and a detailed derivation of the multi-scale material model can be found in Klatt and Haberman (2013).

Finite element (FE) based representative volume element (RVE) homogenization is performed at the microscale to determine the effective mesoscopic stiffness of the material inclusions, with some of the components of the effective stiffness tensor possessing negative values by design. The structure-driven mechanical behavior of the negative stiffness elements is indicative of a new class of materials, which have been labeled mechanically transforming metamaterials (MTM) as a result of their bistable effective constitutive behavior (Haberman, Klatt, Wilson, & Seepersad 2012).

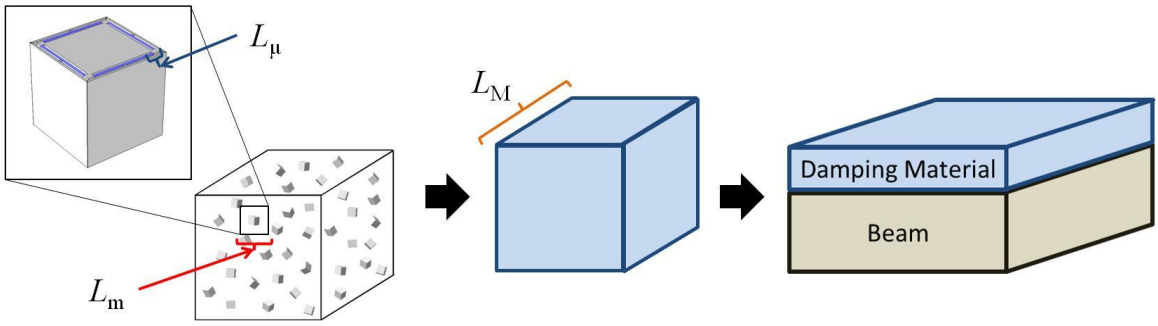

Figure 5.1. An illustration of the hierarchical levels of the composite material, with micro-, meso-, and macro-scales indicated by the subscripts μ, m, and M, respectively. The negative stiffness inclusion is illustrated in the upper left, and the beam coating application is illustrated on the right.

At the mesoscale, effective medium theories are used to determine the effective behavior of a composite material consisting of a homogeneous matrix material containing negative stiffness inclusions. Mesoscale models consider the effective stiffness of the inclusions, as well as their geometry, orientation, and volume fraction to determine the effective stiffness and loss factor of the composite material. It is ultimately the microstructure that leads to mesoscopic negative stiffness behavior and significant increases in absorptive capacity on the macroscopic scale (Klatt & Haberman, 2013).

The effective stiffness and loss properties predicted by the mesoscale models serve as input to a macroscopic model for a layered, composite beam application. A layered beam is coated with the composite material containing negative stiffness domains, and layered plate models from composite beam theory are used to predict the overall stiffness and loss properties of the beam (Ross, Ungar, & Kerwin, 1959). This application investigates the ability of microstructural changes to influence the damped vibration response of a composite beam. The goal is to significantly increase the loss factor of the beam without decreasing its effective stiffness upon application of a layer of negative stiffness composite material.

The design of composite materials with microscale, negative stiffness inclusions is challenging from several perspectives. First, the design space is complex. The full multiscale material model must consider the geometry and material properties of the microstructure as well as the volume fraction, morphology, and orientation distribution of the MTM inclusions at the mesoscale, and the relationships between these variables are sometimes highly nonlinear. Furthermore, computationally expensive FE methods are

required to model the constitutive mesoscale properties of the MTM, and these models are difficult to automate. Similarly, some micromechanical effective medium approaches for modeling the meso- to macro-scale transition can display numerical instability, which prevents full automation of the design exploration process. In addition, the design space includes discrete variables, such as alternative topologies of the MTM inclusions, and highly nonlinear relationships between the geometric characteristics of those inclusions and the macroscopic loss behavior. Uncertainty also plays an important role, for example, in the impact of process-induced variations in the geometry of MTM inclusions on their mesoscale behavior. Though this variation is not explored in this study, it is a nontrivial component of the design of these types of materials and the modeling and design strategy must therefore be of appropriate generality to consider fabrication. All of these characteristics, plus the inherently multilevel nature of the design problem, motivate the need for a comprehensive approach for multilevel design of these material systems.

The hierarchical modeling and design process followed in this Chapter is outlined in the flowchart shown in Figure 5.2. The inclusion geometry parameters are the design variables for the micro- to meso-scale model. The outcome of the micro- to meso-scale model is the inclusion's effective stiffness tensor, which represents the performance variables for this level. The inclusion's effective stiffness tensor is then combined with several matrix properties, as shown in Figure 5.2 to constitute the meso- to macro-scale design variables. The meso- to macro-scale model then implements a homogenization technique to approximate the composite material's effective stiffness and loss factor. The composite metamaterial's effective stiffness and loss represents the performance

variables in the meso- to macro- model, and are then introduced as design variables for a beam coating study, along with the beam parameters shown in Figure 5.2. The composite coated beam is then subjected to static and dynamic analyses to determine the composite's effective stiffness, loss factor, and damping ratio, which represent the beam coating study's performance variables. The consistent transition of lower-level performance variables to upper-level design variables makes this hierarchical design problem challenging, and thus provides an excellent case study to demonstrate the effectiveness of the Bayesian network classifier introduced in previous Chapters.

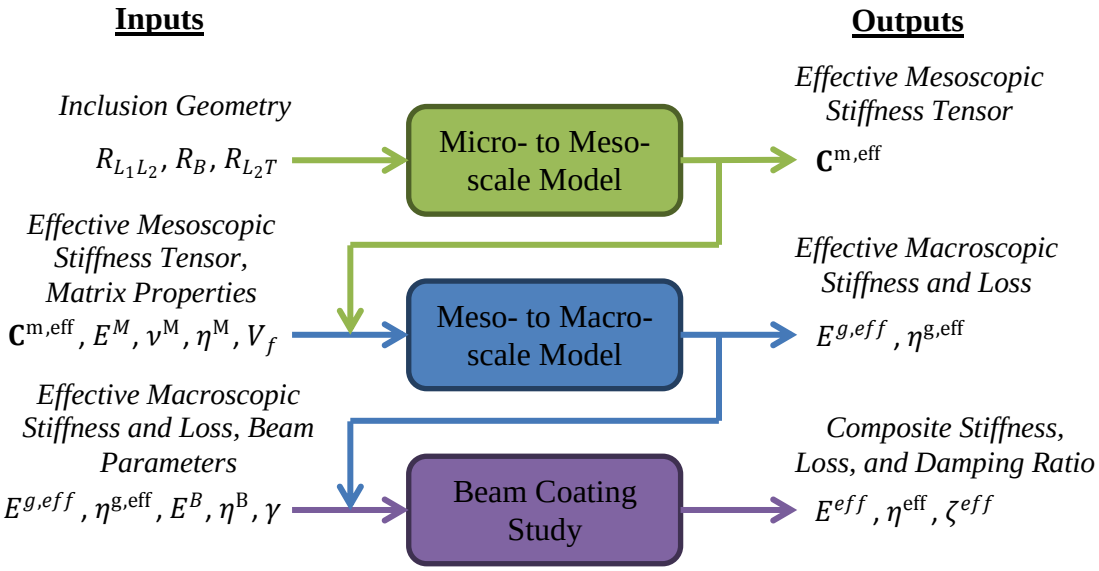

Figure 5.2. Flowchart of hierarchical modeling including variable inputs and outputs at each level.

The micro- to meso-scale and meso- to macro-scale models are presented in Sections 5.2 and 5.3, respectively, along with accompanying design space maps derived

from BNCs. The results from Sections 5.2 and 5.3 are then used to design a coated beam in Section 5.4.

5.2 Micro- to Meso-Scale Modeling and Design Space Mapping

In the micro- to meso-scale model, the geometries of the inclusions are designed to provide negative stiffness in at least one direction. The parameterization of a candidate inclusion and the homogenization approach for predicting its properties is described in Section 5.2.1. Mappings of the design and performance spaces are described in Section 5.2.2.

5.2.1 Layout and Modeling of the Negative Stiffness Inclusions

A candidate geometry for the negative stiffness inclusion is depicted in Figure 5.3. The inclusion is intended to be manufactured using a micro co-extrusion process, in which successive extrusions of a green part reduce the inclusion's external and internal dimensions from millimeter to micrometer scale (Kovar, King, Trice, & Halloran, 1997). During the extrusion process, voids are filled with a carbon black material. Post-extrusion, a high-temperature sintering process pyrolizes the carbon black from the extrudate, which leaves voids within the inclusion. The inclusion is comprised of two materials—alumina and yttria tetragonal zirconia polycrystals (YTZP)—whose coefficients of thermal expansion differ. Differential contraction during the sintering process axially compresses the four alumina beams in Figure 5.3 so that they assume the outwardly buckled, bistable states depicted on the right of Figure 5.3. When the inclusion is embedded in a matrix material and mechanically loaded, the negative stiffness

inclusions lead to greater localized strains, and hence greater stiffness and damping, than positive stiffness inclusions.

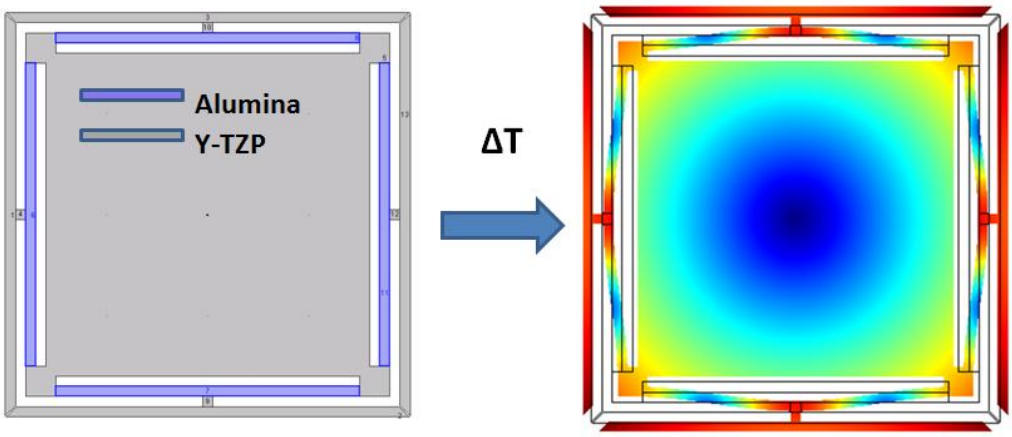

Figure 5.3. A candidate MTM inclusion design showing FE modeling.

Figure 5.4 illustrates the geometric parameters that define the inclusion. The external boundary of the inclusion, and the mesoscale, is defined by the parameter L. The parameter $2H$ defines the height of a T-shaped interface which transfers loading from the surrounding matrix to the center-point of the buckled element. The parameter T defines the width of the connection between the interface and the buckled element. The parameter L_1 locates the midpoint of the inclusion, $\frac{L}{2} - 2H$. The parameter B defines the height of the buckled element and the voided region below it. Finally, L_2 defines the buckled element length. To reduce the dimensionality of the problem, non-dimensional ratios are used to adjust the inclusion geometry. The ratios of interest are $R_{L_1L_2}$, R_B, and R_{L_2T}, as defined by Equations 5.1-5.3.

$$R_{L1L2} = L2/L1 \quad (5.1)$$

$$R_B = 2B/(L1 - L2) \quad (5.2)$$

$$R_{L2T} = T/L2 \quad (5.3)$$

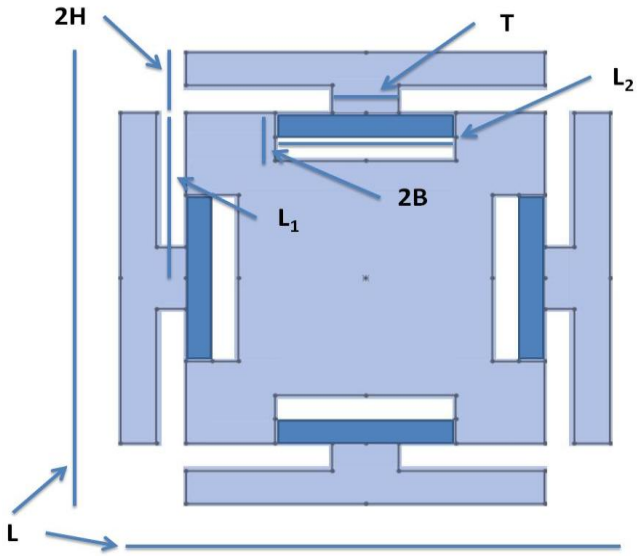

Figure 5.4. Cross-sectional view of parameterized inclusion geometry with all critical parameters. YTZP shown in light blue, with Alumina elements in dark blue.

To relate the geometric parameters of the inclusion to the effective elastic properties of the inclusion, a two-step, nonlinear, multi-scale material homogenization method, similar to that suggested by Odegard (2004), is applied to the inclusion is applied to the inclusion. Nonlinear finite element analysis is used to simulate the force-displacement behavior of the structured inclusion for a series of boundary conditions. The

nonlinear finite element model considers both the geometric nonlinearity of the inclusion structure and the loading induced by differential thermal contraction during the sintering process. The methodology then assumes that the nonlinear stress and strain behavior resulting from the inclusion structure can be well-represented as a continuous elastic solid inclusion with nonlinear mesoscopic effective elastic properties, $\mathbf{C}^{\mathrm{m,eff}}(\mathbf{E}^{\mathrm{m}})$, where $\mathbf{C}^{\mathrm{m,eff}}$ is the effective nonlinear mesoscopic stiffness tensor and $\mathbf{E}^{\mathrm{m}}$ is the mesoscopic Green's strain tensor evaluated on the boundaries of the MTM element. Elements of the effective nonlinear mesoscopic stiffness tensor are calculated from the curvature of the inclusion's strain energy versus Green's strain relationship, using energy methods. The mesoscopic stiffness tensor is then used as a strain-dependent input to the meso- to macroscale transition model in Section 5.3.

5.2.2 Mapping the Micro- to Meso-scale Design Space

The micro- to meso-scale design problem has three design variables, corresponding to the dimensionless parameters in Equations 5.1-5.3, and six performance parameters, corresponding to elements of the mesoscopic effective stiffness tensor, $\mathbf{C}^{\mathrm{m,eff}}$. The performance parameters of particular interest in this example are two of the elements of the stiffness tensor, $C_{11}^{\mathrm{m,eff}}$ and $C_{12}^{\mathrm{m,eff}}$, which are directly related to the plane-strain bulk modulus of the inclusions. Values of the performance parameters are divided into two classifications. The high performance class captures designs that exhibit negative stiffness in the first principal direction: a negative value for $C_{11}^{\mathrm{m,eff}}$. Low performance designs do not exhibit negative stiffness in the first principal direction. Data

from the finite element-based homogenization procedure serve as training points for Bayesian network classifiers of the design and performance spaces, with resulting maps illustrated in Figures 5.5 and 5.6. The training points were defined as the first 1,000 points of the Halton sequence (Freeman & Halton, 1951), interpolated to span the bounds of each non-dimensional geometric ratio specified in Table 5.1.

Table 5.1. Inclusion geometric parameter bounds.

Geometric Ratio	Minimum	Maximum
$R_{L_1L_2}$	0.60	0.99
R_B	0.10	0.98
R_{L_2T}	0.05	0.20

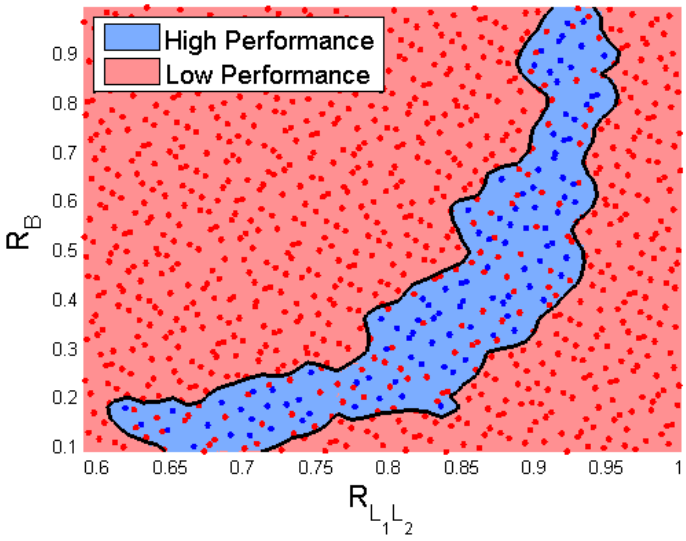

Figure 5.5. Micro- to meso-scale design space mapping.

A two-dimensional plot of the micro-scale design space in Figure 5.5 maps the high performance design points that exhibit negative values for $C_{11}^{\mathrm{m,eff}}$. High (low) performance points are blue (red), and the black line represents the decision boundary between high and low performance designs. Some of the points within the high performance decision boundary are low performance points, indicated by the red points within the blue region. These points appear to be falsely classified because the three dimensional design space has been reduced to two dimensions in this plot. The R_{L_2T} design variable has only a minor influence on $C_{11}^{\mathrm{m,eff}}$, but changes in its value account for the apparent misclassification of points in Figure 5.5, which could be eliminated with a three-dimensional plot.

The high performance points in Figure 5.5 relate to designs in which the four alumina beams shown in Figure 5.3 buckle due to the induced strains during the thermal contraction of YTZP. The absence of high performance points in the upper left quadrant of Figure 5.5 indicates that the induced strain on the beams is insufficient to cause buckling, and supports the notion that a short and thick beam will not buckle. Although not plotted in Figure 5.5, the design variable R_{L_2T} controls the width of the connector between the alumina beam and the YTZP. This parameter has been bounded to prevent inaccurate point-loading simulation results at the lower bound and to prevent detrimental effects to the beam buckling characteristics at the upper bound; otherwise, it has little effect on the performance of the inclusion.

To test the accuracy of the BNC mapping, the classifier in Figure 5.5 was retrained, and the decision boundary was redrawn after omitting the last 10% of the

Halton sequence of design points. Then, those omitted points were classified by the retrained classifier, and their classifications were compared to simulated levels of performance. The classifier correctly classified 89% of all the training points, but misclassified 52% of the high performance points as low performance. For this exploratory design space mapping, the classifier is intended to capture the entire high performance space, and err on the side of misclassifying low performance points as high performance. Adhering to this goal, the high performance class loss factor ratio, λ_1, was increased from unity to 6, resulting in correctly classifying 84% of all the training points, and 89% of the high performance points. This high level of accuracy is significant, given that only 11% of the design space is classified as high performance, and the boundary of that high performance region is very irregularly shaped. Those features make it very difficult to map the high performance region with other approaches, such as interval-based classifiers, which would result in either overly conservative or overly liberal classification of points.

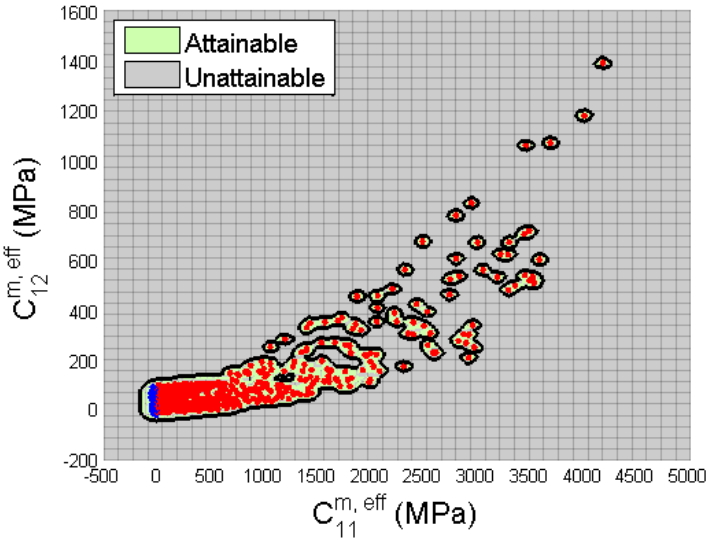

Figure 5.6. Micro- to meso-scale performance space mapping.

All of the design points in Figure 5.5 map to performance points in Figure 5.6, with high performance designs again represented by blue dots. The black lines in Figure 5.6 represent the boundary of the attainable space, which represents the combinations of performance parameter ($C_{11}^{\mathrm{m,eff}}$ and $C_{12}^{\mathrm{m,eff}}$) values that correspond to feasible combinations of design variable ($R_{L_1L_2}$, R_B, R_{L_2T}) values. The micro- to meso-scale performance space mapping shown in Figure 5.6 makes it evident that only a small fraction of the design space maps to high performance designs (shown in blue), with attainable $C_{11}^{\mathrm{m,eff}}$ values ranging from 4,000 MPa in positive stiffness to only -80 MPa in negative stiffness.

When exploring the multi-level design problem, this performance space mapping is intersected with the design space mapping for the meso- to macro-scale model, thereby identifying points that are simultaneously attainable at the micro- to meso-level and linked to satisfactory performance at the meso- to macro-level. For this purpose, it is important to have an accurate mapping of the performance space. The parameter β controls the region of influence of each attainable point, as described in Chapter 3, where a smaller value of β pulls the decision boundary closer to the known attainable design points, reducing the probability of falsely classifying an untested point as attainable but also increasing the probability of falsely classifying an untested point as unattainable. The attainable probability threshold for the performance space mapping shown in Figure 5.6 was determined by setting β to 3. By applying the same cross-validation technique as

in the design space, the classifier trained with 90% of the training points was found to correctly classify 95% of the omitted points, and 100% of the omitted points in the high performance space ($C_{11}^{\mathrm{m,eff}} < 0$). The mapping of the attainable high performance space was optimized by finding the smallest value for β such that the cross-validation accuracy of the high performance points remained at 100%. The optimization resulted in a β value of -0.45, and is illustrated in Figure 5.7. In this mapping, a negative value of β indicates that the high performance training points are clustered closer together than the training points as a whole.

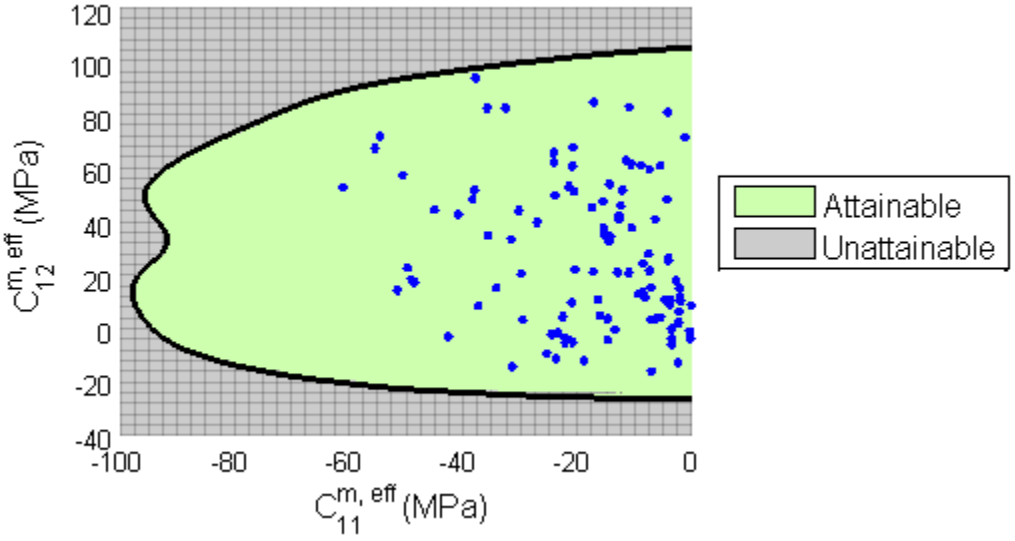

Figure 5.7. Micro- to meso-scale high performance space mapping.

5.3. Meso- to Macro-scale Modeling and Design Space Mapping

In the meso- to macro-scale model, the effective stiffness and effective loss factor are modeled for a composite material, consisting of the negative stiffness inclusions

modeled in the preceding section, dispersed within a viscoelastic matrix. As described in Section 5.3.1, effective medium theory is used to model the effective stiffness and loss properties of the composite. Mappings of the design and performance spaces are described in Section 5.3.2.

5.3.1 Modeling the Performance of the Composite with Negative Stiffness Inclusions

The inputs to the meso- to macro-scale model are the volume fraction, morphology, and orientation of the inclusions within the viscoelastic matrix; the material properties of the matrix; and the effective mesoscale stiffness tensor for the inclusion, quantified with the FEA approach described in Section 5.2.1. Micromechanical effective medium theory (EMT) is used to predict the macroscale effective stiffness and loss properties of the homogenized composite material. EMT is a very general modeling approach for estimating quasi-static macroscale stiffness and loss behavior of viscoelastic composite materials (Haberman *et al*., 2006). Figure 5.8 depicts the meso- to macro-scale transition used in this work. The mesoscale RVE containing a matrix with stiffness $\boldsymbol{C}^{\mathrm{M}}$, an inclusion with stiffness $\boldsymbol{C}^{\mathrm{I}}$, and an inclusion coating with stiffness $\boldsymbol{C}^{\mathrm{C}}$ is homogenized into a macroscale element with stiffness $\boldsymbol{C}^{\mathrm{g,eff}}$.

Using these traditionally static models, stiffness and loss behavior of the composite under dynamic loading can be modeled via the elastic-viscoelastic correspondence principle, as long as the inclusions remain much smaller than the wavelengths induced in the matrix material by dynamic loading (Milton, 2002). The elastic-viscoelastic correspondence principle states that the stiffness tensor of the

homogenized composite can be represented with complex valued entities according to Equation 5.4:

$$\mathbf{C} = \mathbf{C}' + j\mathbf{C}'' = \mathbf{C}'(\mathbf{I} + j\boldsymbol{\eta}) \quad (5.4)$$

Where C is the complex stiffness tensor consisting of storage, $\mathbf{C}'$, and loss, $\mathbf{C}''$, components, I is the identity tensor, and $\boldsymbol{\eta}$ is the loss factor tensor. Under this principle, the usual operations applicable to EMT hold and the overall absorptive properties of the composite can be estimated (Haberman *et al*., 2006).

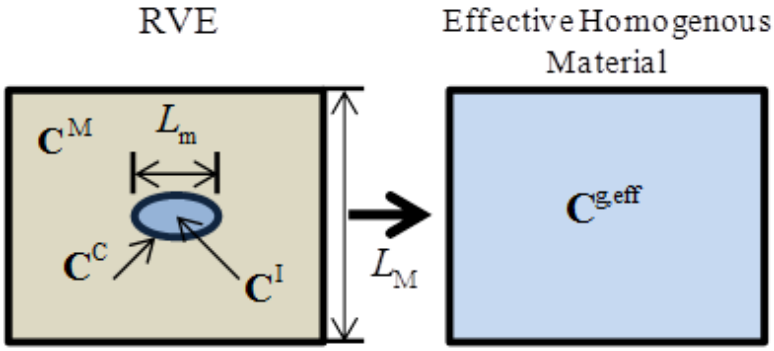

Figure 5.8. Conceptual schematic of the homogenization approach of the Self-Consistent micromechanical model.

In this study, the orientation averaged composite stiffness is determined using a 0.2% volume fraction of inclusions relative to the matrix material. The relevant matrix material properties used in this study are given in Table 5.2.

Table 5.2. Meso- to macro-scale modeling parameters.		
Parameter	Variable	Value
Matrix Young's Modulus	E^{M}	30 MPa
Matrix Poison's Ratio	ν^{M}	0.3
Matrix Loss Factor	η^{M}	0.005
Inclusion Volume Fraction	V_f	0.2%

5.3.2 Mapping the Meso- to Macro-scale Design Space

The meso- to macro-scale model's design variables include the $C_{11}^{\mathrm{m,eff}}$ and $C_{12}^{\mathrm{m,eff}}$ elements of the effective mesoscale stiffness tensor, $\mathbf{C}^{\mathrm{m,eff}}$, for the inclusion; these two elements are assumed to have the largest performance impact on macroscale performance. The model has two performance variables, namely, the effective stiffness, $E^{g,eff}$, and effective loss factor, $\eta^{g,eff}$, of the composite material. The effective stiffness and effective loss factor are defined by the real and imaginary component of the complex valued stiffness tensor of the homogenized composite described in Equation 5.4.

The mapping of the meso- to macro-scale model classifies the design and performance spaces into high performance, low performance, and unattainable regions.

High performance designs provide a loss factor greater than twice that of the matrix material. For the meso- to macro-scale mapping, 1,200 training points are defined by cascading the Halton sequence sampling of the micro- to meso-scale model. Points are evaluated by the meso- to macro-scale model only if they are classified as high performance by the micro-to meso-scale classifier. Using the micro- to meso-scale classifier to filter the training point selection resulted in a 78% reduction in the number of points cascaded to the meso- to macro-scale and limited them to only those points with negative $C_{11}^{\mathrm{m,eff}}$ values.

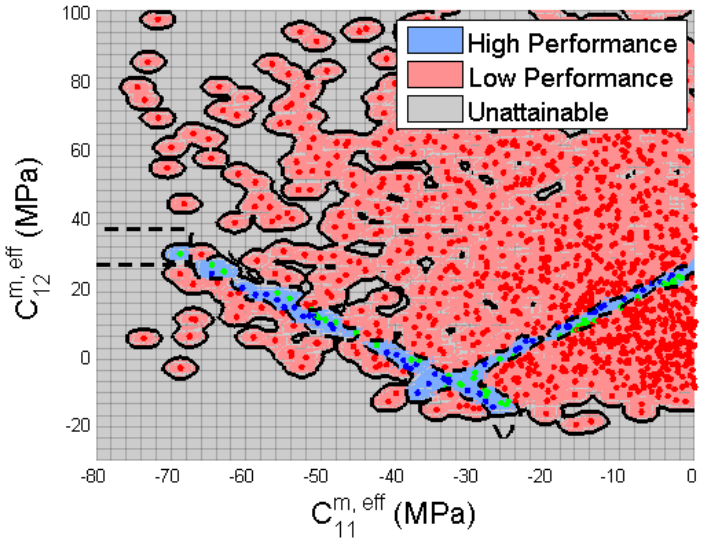

Figure 5.9. Meso- to macro-scale design space mapping.

The meso- to macro-scale design space mapping is illustrated in Figure 5.9, with the attainable probability threshold defined by a β of 3. The solid black lines indicate the boundary of the attainable space, and the dotted black lines indicating the high/low performance class boundary. Within that attainable space, the red dots indicate low

performance designs that do not meet the loss factor threshold. The blue and green dots represent designs with high performance, with the blue dots also meeting a minimum threshold for effective stiffness of the homogenized composite (specifically, the effective stiffness of the composite must be greater than or equal to that of the matrix material). The design space mapping illustrates the narrow band of designs that provide high macroscopic performance, wherein only 5% of the meso- to macro-scale training points are classified as high performance.

To test its accuracy, the meso- to macro-scale design space classifier was reclassified with 90% of the training points and accurately classified 76% of the omitted high performance test points, 85% of the omitted low performance test points, and 84% of all the omitted test points. Accuracy could be improved by increasing the number of training points.

Further analysis of the high performance design space of the meso- to macro-scale model demonstrates intriguing trends. Setting the $C_{11}^{\mathrm{m,eff}}$ stiffness value equal to -20 MPa and unilaterally progressing along the $C_{12}^{\mathrm{m,eff}}$ stiffness value yields the plot shown in Figure 5.10. The bell-shaped normalized loss curve ($\eta^{\mathrm{g,eff}}/\eta^{\mathrm{M}}$) in Figure 510 is to be expected, as the loss factor increases as the design gets closer to its 'sweet spot'. The stiffness curve ($E^{\mathrm{g,eff}}/E^{\mathrm{M}}$) in Figure 5.10 is surprising in that it decreases prior to the 'sweet spot' and increases afterwards–a trend that has been shown by Lakes (2001). Knowledge of these trends aids designers as they search for designs with a combination of high loss and high stiffness.

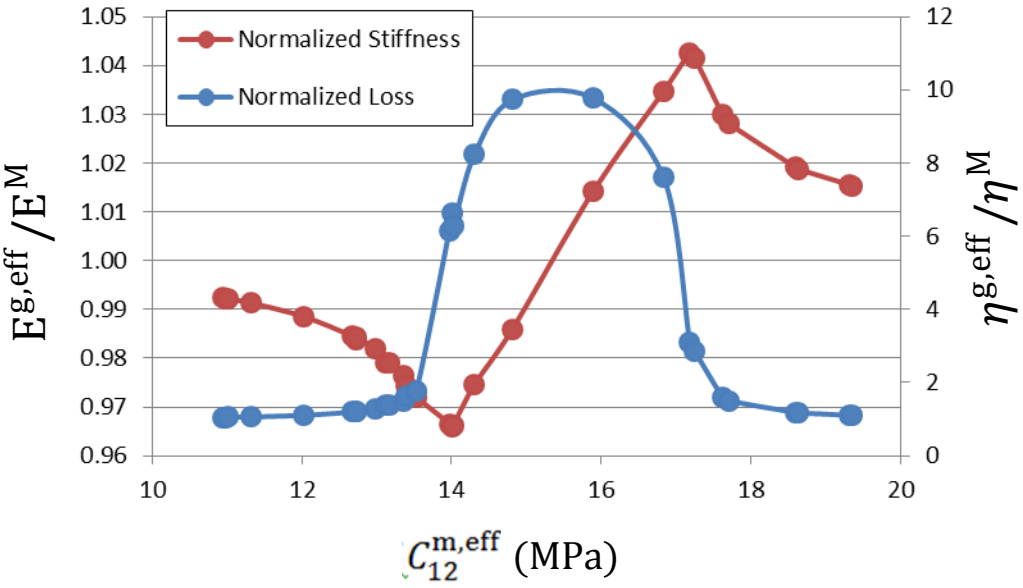

Figure 5.10. The effective stiffness and loss ratio of the composite as a function of $C_{12}^{m,eff}$, with $C_{11}^{m,eff} = -20$ MPa.

The narrow high performance bands shown in Figure 5.9 indicate a linear relationship between $C_{11}^{m,eff}$ and $C_{12}^{m,eff}$ for high performance designs. This trend is explicitly shown in Figure 5.11. The trend line relates the high performance space to a constant sum of $C_{11}^{m,eff}$ and $C_{12}^{m,eff}$. Noting the relationship between $C_{11}^{m,eff}$ and $C_{12}^{m,eff}$ and the plane-strain bulk modulus, $K_{p\varepsilon}$, given by Equation 5.5, these results suggest that the high performing designs exhibit a target value for the plain strain bulk modulus which yields desirable overall combinations of stiffness and loss. It is very interesting to note that the results shown in Figures 5.10 and 5.11, echo results previously published by Lakes (2001), which indicate that specific ratios of inclusion to matrix shear moduli result in drastic increases in the loss factor of the composite.

$$K_{p\varepsilon} = \frac{C_{11} + C_{12}}{2} \qquad (5.5)$$

The y-intercept of the linear trend shown in Figure 5.11 is specific to the particular matrix stiffness chosen in this experiment. One observes that the intercept essentially determines the plane-strain bulk modulus of the inclusion that yields significant increases in the overall lossy behavior of the composite. For this particular case, the ratio of the plane-strain bulk modulus of the inclusion to the bulk modulus of the matrix is approximately -0.02. This ratio may be a useful target for enhanced performance in future studies, although the matrix Poisson's ratio and anisotropy likely influence this value. In general, however, these results provide very interesting insight into the relationship between the plain strain bulk modulus of MTM inclusions and matrix stiffness, which can give designers an intelligent search strategy to find a high loss design for a different matrix material.

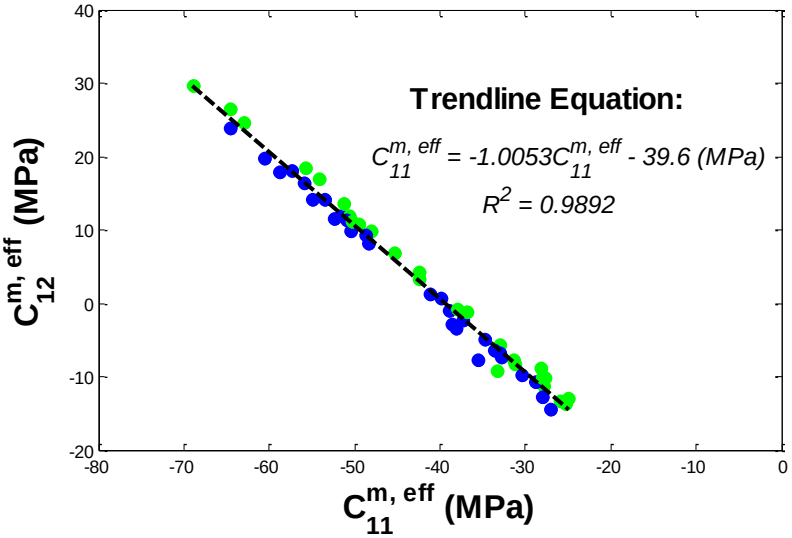

Figure 5.11. A trend line of the negative sloping meso- to macro-scale design space.

The performance map for the meso- to macro-scale model is illustrated in Figure 5.12. The colors of the design points correspond to those in the design space map in Figure 5.9. The macro-scale performance space exemplifies the benefits of loss and stiffness gained in the high performance class, as well as the small fraction of points that fall into this class. The attainable probability threshold is defined by setting β to 10, which captures all of the high performance training points within its attainable space. For this mapping, a high value of β is necessary to accurately capture the high performance space, which has 10x fewer training points than the low performance class, and covers a 10x larger range of normalized loss factor. The classifier was retrained with 90% of the training points, and found to accurately classify 97% of the omitted training points. With respect to high and low performance class individually, the low performance class had much higher accuracy; the retrained classifier accurately classified 99% of the omitted low performance points, while accurately classifying 74% of the omitted high performance points.

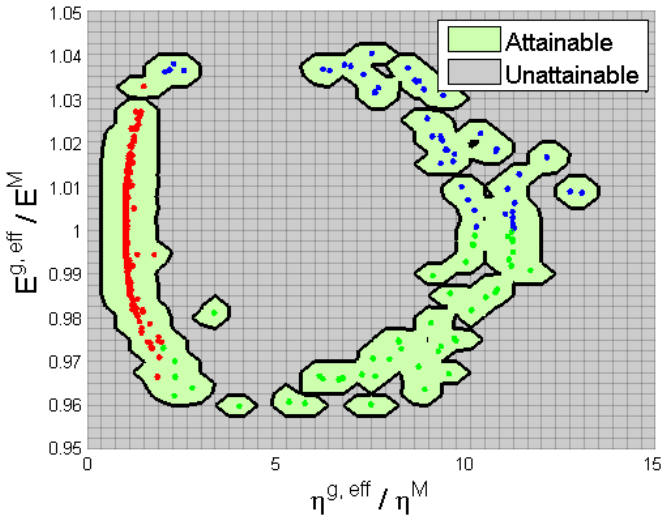

Figure 5.12. Macro-scale performance space mapping.

The next step is to utilize the micro- to meso-scale and meso- to macro-scale mappings to design a component. This step involves intersecting the smaller-scale performance space mapping with the larger-scale design space mapping to identify multiscale sets of designs that achieve a set of performance requirements for the component. Section 5.4 applies the knowledge of the hierarchical design and performance spaces to the design of a composite coating for passive damping of a structural beam.

5.3.3 Backpropagating the BNC Mappings

The preceding sections followed a forward, bottom-up modeling and sampling approach to populate the BNC design and performance space mappings at each hierarchical level. These mappings provide an intuitive understanding of the combinations of design variables that yield satisfactory, high performance values at each level, as well as the ranges of performance values that can ultimately be achieved. The

next step is to utilize the micro- to meso-scale and meso- to macro-scale mappings to design a component, in an inductive, top-down approach. This involves backpropagating the high performance regions in the meso- to macro-scale performance space down to the micro- to meso-scale design space.

The first step to developing this top-down design strategy is to intersect the attainable micro- to meso-scale performance space with the high performance meso- to macro-scale design space. This mapping is shown in Figure 5.13, with the attainable micro- to meso-scale performance space shown in green, and the high performance meso- to macro-scale design space shown in blue.

The intersected high performance region shown in Figure 5.13 is then used to redefine the high performance micro- to meso-scale class, which was previously defined as designs that merely have negative stiffness in the first principle direction. The micro- to meso-scale BNC mapping is then retrained using this updated, stricter performance requirement to determine the micro-to meso-scale design variable combinations that yield macroscale high performance values. The retrained micro- to meso-scale design space mapping is shown in Figure 5.14, with the original high performance micro- to meso-scale design space shown in light blue and the new high performance micro- to meso-scale design space shown in dark blue.

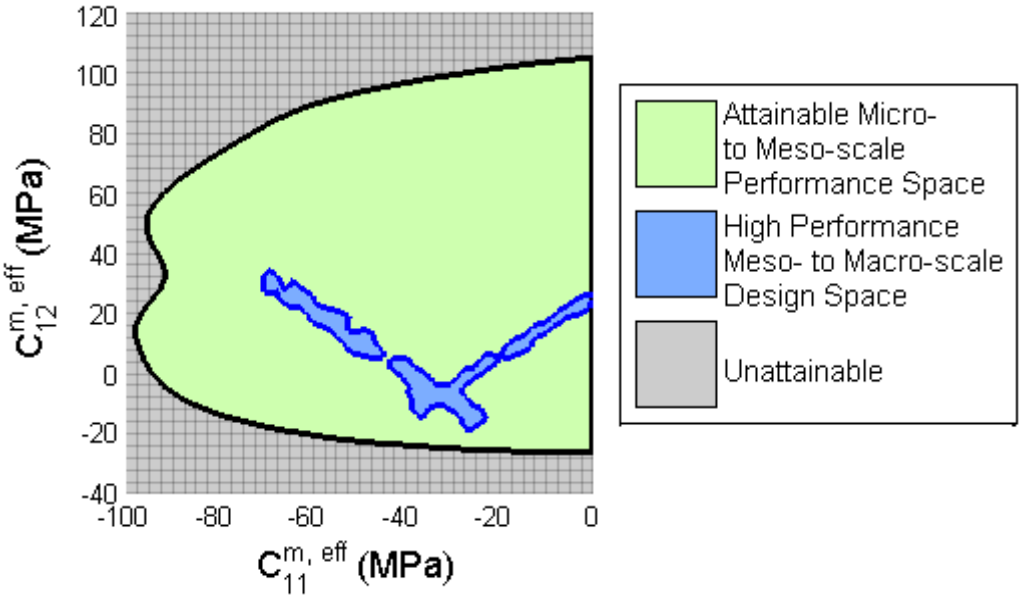

Figure 5.13. The high performance meso- to macro-scale design space intersected with the attainable micro- to meso-scale performance space.

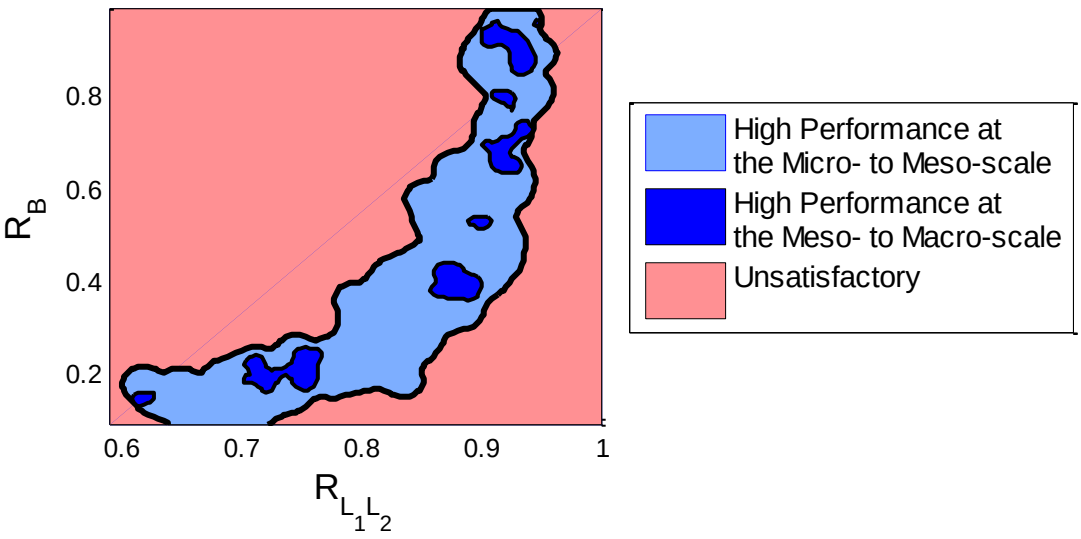

Figure 5.14. The micro- to meso-scale design space mapping with backpropagated performance requirements from the meso- to macro-scale.

The micro- to meso-scale design space shown in Figure 5.14 is used to identify multi-scale sets of designs that meet top-level performance requirements. The backpropagating of the multi-scale design and performance maps in this section is computationally efficient in that it does not require any new concept evaluations to be performed, and can easily be repeated if changes are made to the meso- to macro-scale performance requirements. Section 5.4 applies this multi-scale knowledge to design a composite coating for passive damping of a structural beam.

5.4. Beam Coating Design Case Study

In this section, the design and performance space mappings from Sections 5.2 and 5.3 are used to design a multilayer, cantilever beam with the composite metamaterial as a coating layer for the beam. Section 5.4.1 describes the model for passive viscoelastic damping in a multilayer beam. Section 5.4.2 explores the impact of the design of the composite metamaterial on the beam's shock response.

Throughout this section, two different coatings are investigated—a high performance coating and a low performance coating—to illustrate the benefit of the classifier-based approach in Section 5.3. The high performance coating corresponds to a design on the border of the blue and green points of Figures 5.9 and 5.12, while the low performance coating corresponds to a red design point in Figures 5.9 and 5.12. Specifically, the selected high performance design has an effective coating loss factor, η^{C}, of 0.060, and the low performance design has an effective coating loss factor of 0.005. The high and low performance designs both have equal magnitude effective

stiffness, E^{C}, of 33 MPa. The underlying beam is assigned a Young's modulus, E^{B}, of 1,000 MPa and is assumed to be an idealized lossless material.

5.5.1 Loss Factor and Stiffness of a Multilayer Cantilever Beam

Figure 5.15 illustrates a structural beam coated with a viscoelastic material with microscale inclusions. Ross, Kerwin, and Ungar (Ross et al., 1959) provide a relationship between the loss factor of the composite beam, η^{eff}, and that of the viscoelastic coating material, η^{C}, according to Equation 5.6.

$$\frac{\eta^{\mathrm{eff}}}{\eta^{\mathrm{C}}} = \left[1 + \frac{k^2\left(1+\eta^{\mathrm{C}2}\right) + \left({}^{r_1}/_{H_{12}}\right)^2 a}{k\left(1+\left({}^{r_2}/_{H_{12}}\right)^2 a\right)}\right]^{-1} \quad (5.6)$$

Where $a = (1+k)^2 + (\eta^{\mathrm{C}}k)^2$, H_{12} is the distance between neutral axes of the two components, ri is the radius of gyration, $r_i = \frac{H_i}{\sqrt{12}}$, and $k = K_2/K_1$, where $K_i = E_i A_i$, E_i is the real part of the Young's modulus, and A_i is the cross-sectional area of layer i. The height of the coating is related to the height of the beam by the coefficient, γ, according to Equation 5.7.

$$\gamma = \frac{H_1}{H_1 + H_2} \quad (5.7)$$

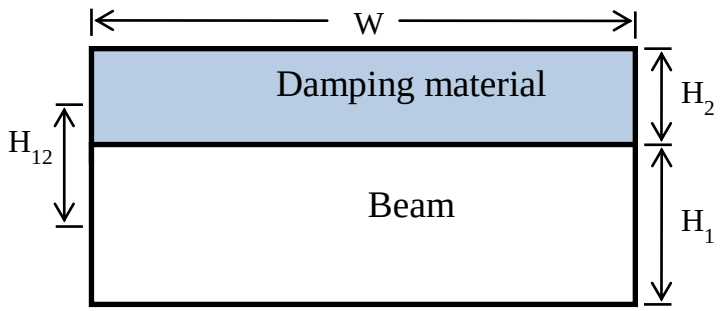

Figure 5.15. Illustration of a beam with a composite coating.

The potential structural benefits of the coating are shown in Figure 5.16, which displays the normalized effective stiffness of the composite beam versus effective loss factor as a function of coating-to-beam height ratio, γ. The effective stiffness, E^{eff}, of the composite beam was calculated using the Voigt approximation (Voigt, 1889), and is normalized by the beam stiffness, E^{B}, in order to demonstrate the effect of the coating on the overall stiffness of the composite beam. The effective loss factor of the composite, η^{eff}, was determined from Equation 5.6.

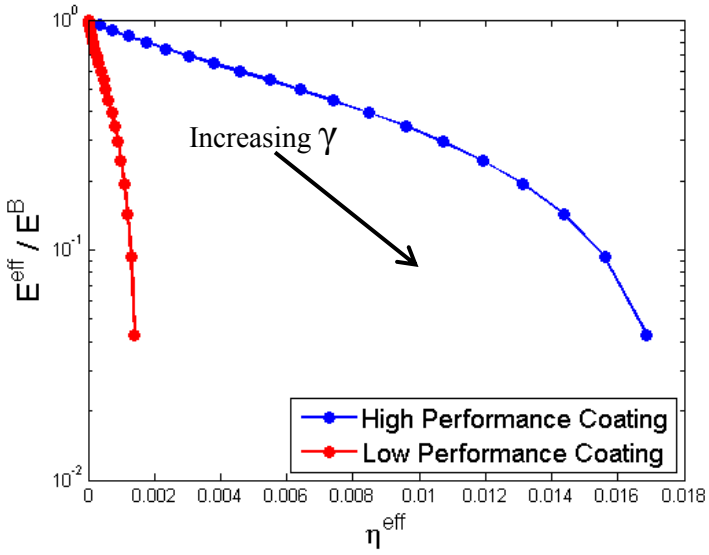

Figure 5.16. Normalized stiffness vs. effective composite loss factor with varying coating thicknesses.

As shown in Figure 5.16, the composite beam's loss factor increases with increasing coating thickness. For the high performance coating, this increase occurs at a much greater rate relative to the low performance coating, while both high and low performance coatings experience equivalent decreases in the overall stiffness of the beam. This trend is expected because the high and low performance designs were chosen to have equivalent effective stiffness but with very different effective loss factors.

A designer can use Figure 5.16 to select a coating-to-beam height ratio based on application specific stiffness and loss requirements. At a coating-to-beam height ratio, γ, of 0.2, the effective composite stiffness is decreased by 19%, with a loss factor, η^{eff}, of the high performance composite equal to 0.0017, and a loss factor of the low performance coating of 0.00014. Similarly, if the coating-to-beam height ratio, γ is increased to 0.5,

the effective composite stiffness of both designs are decreased by 48%, with a loss factor of the high performance composite of 6.4×10^{-3} and a loss factor of the low performance coating of 5.3×10^{-4}.

5.5.2 Transient Response of the Multilayer Cantilever Beam to an Impulsive Load

The coated beam is assembled in a cantilever orientation, as shown in Figure 5.17, and subjected to an impulsive load at its free end. The length, L, and width, W, of the cantilever beam are 1 m and 0.2 m, respectively. The overall cross-sectional height of the beam, H1 + H2, is 0.1 m, and the individual heights of the beam and viscoelastic material are determined by γ in Eq. 8. The density of the beam is 1000 kg/m2, and the density of the viscoelastic material is 200 kg/m2. Although the cantilever beam is an idealized structural component, it helps illustrate the potential value of the negative stiffness composite for vibro-acoustic dampening applications.

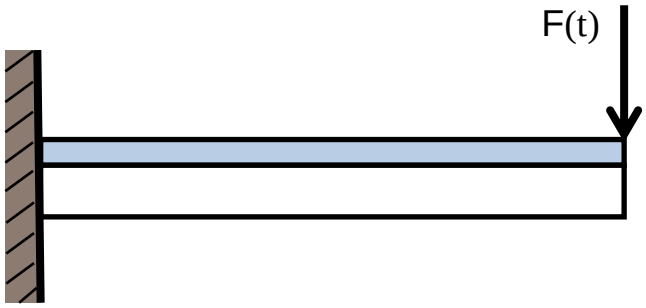

Figure 5.17. An illustration of the coated cantilever beam with an impulsive load imposed at its free end at t = 0.

The coating-to-beam thickness ratio for this demonstration study is chosen based on Figure 5.16 to exhibit one order of magnitude decrease in stiffness over the beam

alone, which corresponds to $\gamma = 0.93$. Although it is not desired to decrease the effective composite stiffness, this coating-to-beam thickness ratio will highlight the high damping capabilities of the coating. For the choice of coating, the loss factors for the high and low performance composites are 0.015 and 0.0013, respectively. For this experiment, the beam tip is subjected to a 10 g magnitude versine shock input with a 25 ms duration starting at t = 0. The resulting tip displacement for two cases of interest is plotted in Figure 5.18.

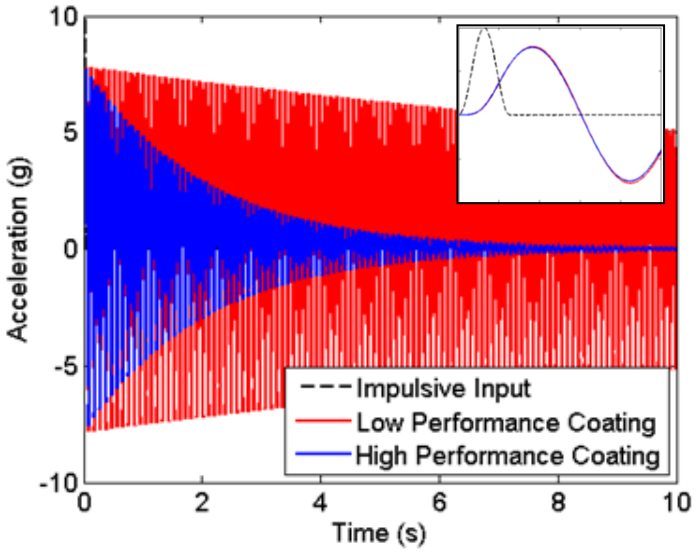

Figure 5.18. The coated and uncoated cantilever beam shock response.

As shown in Figure 5.18, the tip displacements of the two composite beams exhibit oscillating ringdown, because neither composite approaches critical damping. The high performance coating causes the beam to ring down significantly faster than the low performance coating, however, and both are a profound improvement over the underlying beam alone, which would theoretically oscillate forever. The high performance composite

has a damping ratio, $\zeta^{c,high}$, of 0.75, while the low performance composite has a damping ratio, $\zeta^{c,low}$, of 0.065. This is equivalent to a log decrement of -0.047 for the high performance composite and -0.0041 for the poor performance composite. The high performance composite attenuates the acceleration to 0.7% of its initial acceleration amplitude after 10 seconds, while the poor performance coating composite attenuates its acceleration amplitude to only 65% of its initial amplitude during the same time period. The acceleration attenuation of the high performance composite equates to a time constant of 2.04, while the poor performance composite has a time constant of 23.5.

5.6. Discussion

The materials design approach presented in this Chapter uses BNC's for mapping design and performance spaces at each hierarchical level and then intersecting the mappings to identify high-performance system level designs in a top-down, inductive, design strategy. The sampling that is required to create the BNC-based mappings at each level can be performed concurrently across the levels, although creating the models sequentially allows mappings from one level to be used to bound the design or performance space for another level. After sampling is performed at each level, it is almost trivial to identify multi-level designs that meet specific high-level performance requirements by propagating high-performance thresholds from the highest to the lowest level. Characterization of the space of high performance designs at the highest materials level also allows product design to be decoupled from the design of the material.

The multi-scale design approach using BNC mappings significantly reduces the computational expense of the hierarchical materials design example in this Chapter. Capturing the knowledge of the micro- to meso-scale modeling, in which only 11% of designs resulted in a negative stiffness inclusion, allowed the meso- to macro-scale modeling to narrow its candidate design space to only a few feasible designs. The meso- to macro-scale model, for which only 10% of the design space resulted in satisfactory macroscale performance, reinforced the necessity for a top-down cascading of performance requirements. A trial-and-error bottom-up design approach, in which a specific inclusion geometry is propagated through three levels of modeling, would have resulted in over 99% of the analyzed designs having unsatisfactory macroscale performance. Similarly, a purely optimization based approach, such as one of the Multidisciplinary Design Optimization approaches reviewed in Chapter 2, would have required extensive nested iteration and tight coordination between the levels, neither of which is required for the approach proposed here.

The set-based design approach facilitated by the use of BNC maps allowed for the decomposition of this design problem into multiple hierarchical levels. By assigning a conservative performance requirement to the micro- to meso-scale level, design and performance flexibility was maintained. Following this explorative phase of the design process, a designer can then set arbitrary meso- to macro-scale performance requirements and easily backpropagate the new high performance regions down to the micro- to meso-scale design space, as detailed in Section 4.3.3. Further exploration of designs having high meso- to macro-scale performance can be attained by sampling from this

backpropagated micro- to meso-scale high performance design space. A designer can additionally use the design and performance flexibility metrics proposed in Chapter 4 to intelligently narrow the meso- to macro-scale performance requirements.

The micro- to meso-scale level is found to have a design flexibility of 0.22, which is slightly higher than the percent of designs that were found to be classified as high performance (11%). This discrepancy is due to the collapsing of the micro- to meso-scale design space from three to two dimensions, which resulted in some low performance designs being misclassified as high performance. To determine the micro- to meso-scale performance flexibility, the desired micro- to meso-scale performance space is defined by placing a lower and upper bound on $C_{11}^{m,eff}$ of -100 MPa and 0 MPa, respectively, and a lower and upper bound on $C_{12}^{m,eff}$ of -40 MPa and 120 MPa, respectively. The micro- to meso-scale level is found to have a performance flexibility of 0.65, representing the attainable percentage of the desired micro- to meso-scale performance space.

The high performance region of the meso- to macro-scale design space, which corresponds to the attainable micro- to meso-scale performance space, is found to have a design flexibility of 0.022. The meso- to macro-scale desired performance space is defined by placing a lower and upper bound on $E^{g,eff}/E^{M}$ of 0.95 and 1.05, respectively, and a lower and upper bound on $\eta^{g,eff}/\eta^{M}$ of 2 and 15, respectively. The meso- to macro-scale is found to have performance flexibility of 0.079. By backpropagating the meso- to macro-scale high performance requirements down to the micro- to meso-scale design space, as detailed in Section 5.3.3, the micro- to meso-scale is found to have design flexibility of 0.037.

To demonstrate the effect of narrowing the meso- to macro-scale performance requirements on the design and performance flexibility metrics at each hierarchical level, a new meso- to macro-scale performance requirement is defined. This particular meso- to macro-scale performance requirement is selected to isolate and examine the outer ring of high performance values in the meso- to macro-scale performance space, and demonstrates the ability of the BNC to accept nonlinear performance requirements. The new meso- to macro-scale performance requirement is shown by the blue highlighted region in Figure 5.19, with the training points plotted in blue if they meet the new performance and in red if they do not. As a measure of the reduction in the ability to meet the new performance requirements, the meso- to macro-scale performance flexibility is now found to be 0.038, roughly half the value found using the original performance requirements.

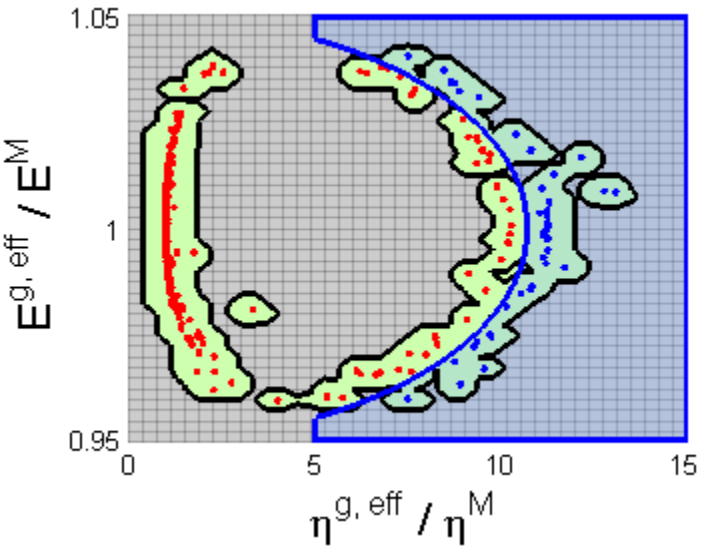

Figure 5.19. Meso- to macro-scale performance space with new performance requirements highlighted in blue.

The BNC is then retrained using the new meso- to macro-scale performance requirement to define the high and low performance classes at each hierarchical level. Figure 5.20 shows the meso- to macro-scale design space mapping, as classified by the new performance requirement. The updated high performance meso- to macro-scale design space has eliminated the positive sloping portion of the high performance design space seen in Figure 5.9. This result affirms that the best designs occur when the microscale inclusion exhibits negative stiffness in its first principle direction. The meso- to macro-scale design flexibility is found to be 0.011, which, similar to the meso- to macro-scale performance flexibility, is roughly half the value found using the original performance requirements. This result indicates that there is a strong correlation between changes in design and performance flexibility.

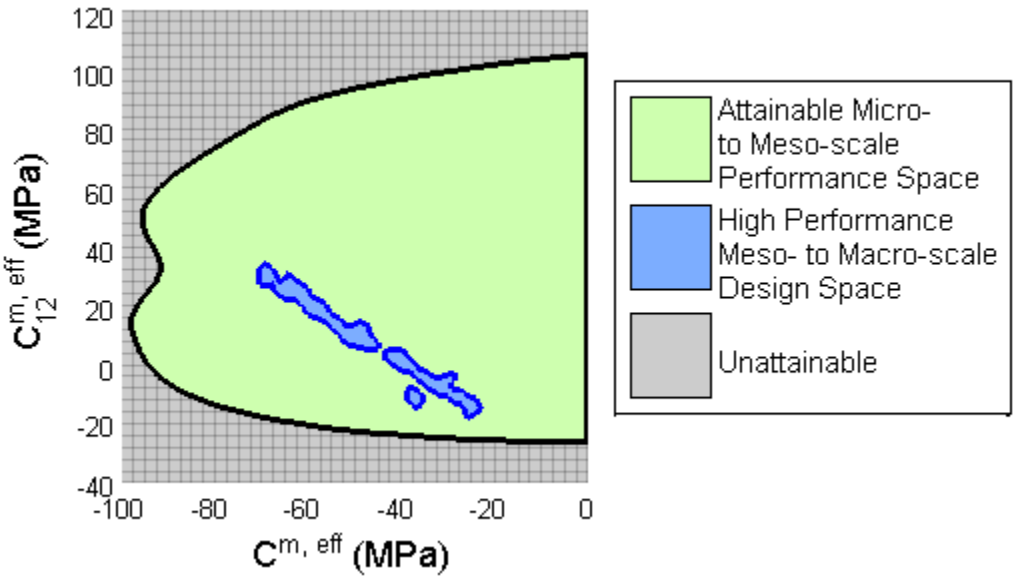

Figure 5.20. Meso- to macro-scale design space reclassified with new performance requirements.

The sets of designs meeting the new meso- to macro-scale performance requirements are then backpropagated down to the micro- to meso-scale design space, shown in Figure 5.21. The narrow regions of high performance designs, backpropagated to micro- to meso-scale design space in Figure 5.21, further demonstrate the nonlinearity of this design problem, and the difficulty that a designer would face in trying to solve this problem using a traditional design approach. By first populating the BNC maps at each hierarchical level, the top-level performance requirements can be easily altered and backpropagated to the bottom-level design space, without any additional concept evaluations required.

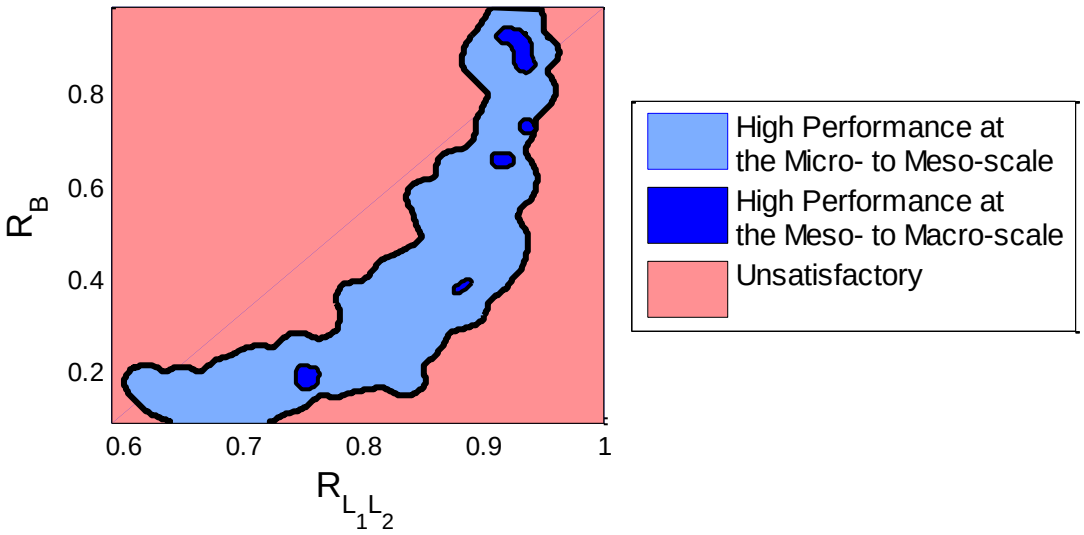

Figure 5.21. The micro- to meso-scale design space mapping backpropagated from the updated meso- to macro-scale performance requirements.

The BNC maps also provided an effective means of capturing the information gained from the multilevel modeling in this example, and allowed the designers to build

intuitive knowledge of the multi-level design space. Designers used the BNC maps to determine where good solutions lie, as well as to discover trends in the design space, as seen in the meso- to macro-scale mappings. The knowledge captured in the BNC maps allowed the beam application to be solved as an independent problem, without requiring additional material modeling. With the aid of BNC maps, choosing a composite coating for the beam application was trivial, as designs could be quickly chosen by selecting a desirable combination of stiffness and loss from the macroscale performance space. The chosen composite coating design could then be backpropagated using the BNC maps to determine the corresponding microscale inclusion parameters.

The effectiveness of the BNC for classifying arbitrarily shaped regions of interest in the design space has been shown in the two-dimensional mapping plots, however the underlying equations are readily extensible to a multivariate problem. Additionally, the probability distributions generated by the BNC mapping can also be used to guide a sampling strategy of the design space, such as to exploit the likely high performance regions of the design space, or to explore regions of the design space that have very little data available. A sampling strategy based on the probability distributions could potentially uncover high performance designs with significantly reduced computational time; however, an effective strategy to progress from exploration to exploitation is a topic for future work.

Chapter 6: Closure

The research presented in previous chapters of this thesis serves to provide a higher fidelity representation of design and performance spaces than the previously proposed methods discussed in Chapter 2. The methods presented are building blocks that improve the ability to achieve a set-based design process, as outlined by Sobek *et al*. (1999), to better equip designers with the tools to work concurrently, rather than sequentially. The method presented uses a Bayesian network classifier, which uses concept evaluations as training points to develop an accurate set representation of the attainable and satisfactory design and performance spaces.

6.1 Summary

The use of a Bayesian network classifier has recently been proposed to create a mapping of the satisfactory design space (Shahan & Seepersad, 2012). The method proposed in Chapter 3 extends this work to create a mapping of the performance space, which has several fundamentally different properties than the design space. While the primary challenge in creating a mapping of the design space is to identify the satisfactory design space, the challenge in creating a mapping of the performance space is to identify the attainable regions that map back to combinations of design variables. This challenge was not encountered in developing the design space mapping because the entire design space can typically be sampled. The bounds on the design space, combined with nonlinearities in the performance models, limit the set of attainable performance values. The method to classify the attainable from unattainable performance spaces utilized a

kernel density estimation technique, which translated the set of training points into a probability distribution spanning the entire performance space, and defined a minimum probability threshold for a performance point to be classified as attainable. The threshold probability can also be used as a tuning parameter to accurately map irregularly distributed spaces.

The attainable performance space is then further classified into a feasible set of designs that meet a set of constraints, and an infeasible set that does not. This classification utilizes the probability distributions returned by the kernel density estimation technique to create a class conditional probability distribution for both the feasible and infeasible classes. The class conditional probability distributions are then used to create a Bayesian network classifier to identify whether an unknown point in the performance space is more likely to be feasible or infeasible. The feasible set of designs in the performance space can also be further narrowed by assigning performance requirements that designs must meet to be classified as feasible. Classifying the performance space into an attainable set, and then further extending the attainable performance mapping into a feasible and infeasible subset provides a complete mapping of the performance space.

Error rate simulations were performed to analyze the classifier's convergence as new training points were added to the classifier, and it was seen that for the two-variable problem examined in Chapter 3, the error converged to less than 5% after 100 training points. This convergence to a low error rate demonstrates the effectiveness of the Bayesian network classifier at representing the design and performance spaces. The

classifier is also computationally efficient, requiring a linear increase in computational expense as the number of training points is increased for a fixed number of variables.

Prior methods to quantify design and performance flexibility were reviewed in Chapter 2, and it was found that the quantification accuracy was directly dependent on the method for set representation. The reliable and accurate mappings provided by the Bayesian network classifier were then utilized in Chapter 4 to quantify design and performance flexibility. The probability distributions returned by the Bayesian network classifier were sampled using a Monte Carlo method to approximate the size of the satisfactory design and performance spaces. This Monte Carlo method of flexibility quantification was shown in Chapter 4 to be both more accurate than interval methods and more computationally efficient. This achievement provides a significant contribution to the set-based design research, which has lacked a consensus on an accurate and effective design and performance flexibility quantification (Ferguson, Siddiqi, Lewis, & de Weck, 2007).

The final contribution of this research was a demonstration of the Bayesian network classifier's mapping of the design and performance spaces, in a multi-level material design problem. The small-scale design of the composite metamaterial involves the geometrical design of a microscopic inclusion, which is then analyzed to evaluate the effective stiffness of the inclusion. The large-scale design takes the effective inclusion stiffness, along with matrix properties, to arrive at a macroscopic stiffness and loss of the composite metamaterial. The Bayesian network classifier allows for designs to be easily propagated from the macroscopic performance space all the way down to the microscopic

design space, without requiring further simulations. The classifier also provides a means to visualize the spaces in two dimensions, and extract trends of the satisfactory design and performance spaces. The use of the Bayesian network classifier in this design problem allows for a reduced number of simulations required to find a set of satisfactory macroscopic parameters, which corresponds to less than 1% of the microscopic design space.

6.2 Future Work

The research presented in this thesis provides a beneficial improvement in the methods necessary to have a fully concurrent set-based design process; however, there are a number of ways that these methods could be improved. The proposed Bayesian network classifier uses a kernel density estimation technique to approximate the class conditional probability distribution, using Gaussian kernels. Future work on this classifier should analyze alternative Gaussian kernel probability distribution parameters, which may better capture the underlying distributions that it attempts to model. Adding additional parameters of kurtosis and skew may lead to increased accuracy; however, these parameters cannot be trivially introduced, and must be combined with a clustering algorithm to vary the parameters within the design and performance spaces. Additionally, alternative kernel probability distributions should be analyzed, such as the lognormal asymmetric probability distribution. Perhaps a library of kernel probability distributions can be drawn from, allowing designers to choose the best kernel to fit their specific problem.

Additional work must be done to analyze the effectiveness of the Bayesian network classifier under a large number of design and performance variables. The "curse of dimensionality" has been well documented in literature, and results from decreasingly small values of the probability distribution, as the dimensionality of a space increases. Varying the assumption of variable dependence has been recently shown to increase the effectiveness of the Bayesian network classifier in mapping a large dimensional space, and should be explored further (Backlund, 2012). Additionally, in regards to the assumption of variable dependence, a method to extract the variable dependence from the Naïve Bayesian network classifier could also be beneficial to this research.

The effectiveness of the Bayesian network classifier can also be improved by intelligently defining the prior probability, which was approximated in this research as a function of the class occurrence frequency of the training points. The prior probability can be used to capture expert knowledge from designers, to give the classifier a better accuracy than the training points alone. Although the expert knowledge may not always be correct, the classifier should be able to mitigate poor assumptions given a large enough set of training points.

The performance space mapping provided by the Bayesian network classifier provides a good approximation of the Pareto frontier, with relatively few training points required. This is a significant achievement, as the Pareto set is typically difficult to define, but can give designers an understanding of the performance potential that exists within a set of designs. Designers always make tradeoff decisions, but they are typically delayed to the end of the design process, when more complete information about the

tradeoff is known. By obtaining an approximation for the tradeoffs that will be encountered, designers can focus more resources in making an optimal tradeoff decision. Future work can focus on utilizing the approximated Pareto frontier to inform decisions in the early stages of the design process.

The set-based design guidelines provided by Sobek *et al*. (1999) state that the design space should be intelligently narrowed. The flexibility quantification methods proposed in this research should next be analyzed to determine an optimal relationship between design and performance flexibility to provide an intelligent narrowing of the design space. This task can be coupled with an informed sampling strategy that uses the probability distributions provided by the Bayesian network classifier to focus the sampling on regions of high potential. This form of guided sampling has recently been proposed by Backlund (Backlund, 2012), and has been shown to perform better than other global search algorithms, such as a genetic algorithm. While the method proposed by Backlund shows high potential, future work is needed to optimize the framework for balancing exploration and exploitation in the guided sampling approach.

Appendix A

The helical spring design problem used in Chapters 3 and 4 is defined in this appendix. This problem has been adapted from Shahan and Seepersad (2012), with a background on the problem found in a machine elements textbook (Juvinall & Marshak, 2000). The helical spring is illustrated in Figure A.1, showing the two design variables, X_1 and X_2, shown as the coil diameter, D, and wire diameter, d, respectively.

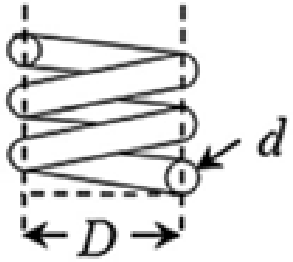

Figure A.1. Diagram of the helical spring.

The design space is confined by upper and lower bounds placed on the design variables, X_1 and X_2, which are defined in table A.1.

Table A.1. Helical Spring Design Variable Bounds.

	X_1 (m)	X_2 (m)
Lower Bound	0.19	0.13
Upper Bound	1.30	0.70

This design problem has two constraints. The first constraint, g_1, defines the maximum allowable shear stress, according to Equation A.1, to not exceed 45% of the ultimate strength of the material under a maximum load, F_s. The loading on the spring is applied at the end of the spring, as illustrated in Figure A.2. The second constraint, g_2, defines the minimum ratio of the coil diameter to wire diameter, according to Equation A.2. In the context of helical springs, the constraint g_2 is referred to as the spring index, and is an important manufacturing consideration.

$$g_1 = \tau_{max} - 0.45S_u = \frac{8F_s X_1}{\pi {X_2}^3}\left(1 + 0.5\frac{X_2}{X_1}\right) - 0.45S_u \leq 0 \quad \text{A.1)}$$

$$g_1 = 3 - \frac{X_1}{X_2} \leq 0 \quad \text{A.2)}$$

The helical spring design problem has two performance variables. The first performance variable, Y_1, represents the maximum allowable compression, L_t, under a target load, F_t. The loading and compression is defined according to Figure A.2. The spring compression, and performance variable Y_1, is defined according to Equation A.3. The second performance variable, Y_2, represents the total width of the spring, and is defined according to Equation A.4. The total spring width is important for this design problem because the spring must be able to fit into a pre-manufactured slot.

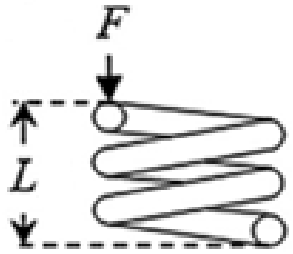

Figure A.2. An illustration of the helical spring loading and compression.

$$Y_1 = L_t = \left(\frac{{X_2}^4 G}{8{X_1}^3 K} + 2\right) X_2 + \frac{F_s - L_t}{K} \tag{A.3}$$

$$Y_2 = w = 1.1D + d \tag{A.4}$$

There is also one performance target for each performance variable. A design is considered to be satisfactory if it meets both performance targets. The performance requirements are defined for Y_1 and Y_2, according to Equation A.5 and A.6, respectively.

$$Y_1 \leq 0.0635 \text{ m} \tag{A.5}$$

$$Y_2 \leq 0.0381 \text{ m} \tag{A.6}$$

There are a set of constants for this design problem, which define the material properties and target loadings. These constants are defined in Table A.2.

Table A.2. Helical spring design problem constants.		
Parameter	*Variable*	*Value*
Spring Stiffness	K	*15,761 N/m*
Target Loading	F_t	*266.9 N*
Maximum Loading	F_s	*513.8 N*
Material Ultimate Strength	S_u	*1.45 GPa*
Shear Modulus	G	*792.9 GPa*

Appendix B

The Matlab® code for developing the Bayesian network classifier (BNC) applied to the helical spring design problem in Chapters 3 and 4 is presented in this appendix. Much of this code is credited to the assistance of Dr. David Shahan. Table B.1 provides a list of the functions included, and a description of what each function does. Basic plotting functions of the design and performance spaces are included, and are modular with their inputs to be able to generate many of the plots included in this thesis.

Table B.1. List of Matlab® functions for creating and using a Bayesian network classifier.

demo_Spring( )	A demo file for using the BNC
kbnInit_Spring( )	Generates an empty BNC structure
systemsKBN_Spring( )	Provides system-level organization for adding training points
kbnAddData_Spring( )	Adds training points to the BNC
kbnEvalDesign( )	Calculates the probability distribution at a design point
kbnEvalPerf( )	Calculates the probability distribution at a performance point
kbnEvalHd( )	Calculates the kernel bandwidth for the design space
kbnEvalHp( )	Calculates the kernel bandwidth for the performance space
kbnEvalPerfUnattP( )	Calculates the attainable performance probability threshold
kbnMonteCarlo( )	Performs a monte carlo integration
plotDSpace( )	Plotting function for the design space
plotPSpace( )	Plotting function for the performance space
halton( )	Generates the Halton sequence

```
function n=demo_Spring(varargin)

% Train the BNC
M=100; % Number of training points
n = struct([]);
for i=1:M
    n = systemsKBN_Spring(n,1); % Collect training points
end

% Set alpha and beta
alpha = .4;
beta = 3;
n.hdalpha = alpha*sqrt(12);
n.hd = kbnEvalHd(n);
n.hpalpha = alpha*sqrt(12);
n.hp = kbnEvalHp(n);
n.hpbeta = beta*sqrt(12);

% Plot the design Space
divs = [100 100];
classAmp = [1 1];
plotDSpace(n,divs,classAmp);

% Plot the satisfactory design space
divs = [100 100];
classAmp = [1 1];
plotPSpace(n,divs,classAmp,'Orientation','2D','LegendLoc','NorthEast',
    ...'DataPoints','off');

% Perform Monte Carlo Estimate of design flexibility
numSamples = 10000;
classAmp = [1 1];
classSat = 1;
classAll = [1,2,3];
MC_sat = kbnMonteCarlo(n,'d',1,numSamples,classAmp);
MC_all = kbnMonteCarlo(n,'d',[1, 2, 3],numSamples,classAmp);
designFlexibility = MC_sat.volume / MC_all.volume;

% Perform Monte Carlo Estimate of performance flexibility
numSamples = 10000;
classAmp = [1 1];
classSat = 1;
classAll = [1,2,3];
MC_sat = kbnMonteCarlo(n,'p',classSat,numSamples,classAmp);
MC_all = kbnMonteCarlo(n,'p',classAll,numSamples,classAmp);
perfFlexibility = MC_sat.volume / MC_all.volume;

end
```

```
function n=kbnInit_Spring(Din, Dout, Dcon, Cd, Cp, dbnd, pbnd, ptarg,
ptargS)
% Initializes the structure for the training points n

n.Din=Din; % # of dimensions of design variables
n.Dout=Dout; % # of performance variables
n.Dcon=Dcon; % # of contstraints

n.Cd = Cd;
n.Cp = Cp;

n.N=0; % Number of training points taken - Initialized to 0
n.Nc=zeros(1,3);    % Number of data points in each general class
n.Ncd=zeros(1,2); % Number of data points in design space classes
n.Ncp=zeros(1,2); % Number of data points in performance space classes

n.dValues=[]; % Holds the training point design variables
n.cValues=[]; % Holds the inequality constraint values
n.pValues=[]; % Holds the training point performance values

n.hd = []; % Kernel standard deviation for design space points
n.hp=[];   % Kernel standard deviation for performance space points
n.stdD=[]; % Standard deviation of design space points
n.stdP=[]; % Standard deviation of performance space points

n.w = []; %weights, N by C
n.wd=[];
n.wp=[];

n.hdAlpha=1;
n.hpAlpha=1;
n.hpBeta = 1;

% Set the boundary of Design Space
n.dbnd = dbnd;
n.dscale = 1./(n.dbnd(2,:)-n.dbnd(1,:));
n.dshift = n.dbnd(1,:);

% Set the approximate boundary of the performance space
n.pbnd=pbnd;
n.pscale = 1./(n.pbnd(2,:)-n.pbnd(1,:));
n.pshift = n.pbnd(1,:);

% Set the approximate boundary of the satisfactory performance space
n.ptarg=ptarg;
n.ptargS=ptargS;

end
```

```
function n=systemsKBN_Spring(n,M,varargin)
% This function creates a KBN classifier, and adds 'M' training points
% to it. Sample points will be chosen from a halton sequence, or from a
% test set inputted into 'varargin'. Note: 'varargin' must include
% entire test set, and the n.N+1 point will be selected for the next
% training point

% Initialize new Network
if isempty(n)
    % Experiment Parameters
    Din=2;
    Dout=2;
    Dcon=2;
    Cd=2;
    Cp=2;
    dbnd_metric=[.0178 .0033; .033 .0048];
    pbnd=[0 .02;.35 .043];
    ptarg=[.0635 .0381];
    ptargS={'min', 'min'};

    % Create training point database
    n = kbnInit_Spring(Din,Dout,Dcon,Cd,Cp,dbnd_metric,pbnd,ptarg,...
        ptargS);
    n.hpAlpha=sqrt(12)*0.4;
    n.hpBeta = 3;
end

% Add M new points to the Network
for i=1:M
    % Sample a new design point
    if isempty(varargin)
        xhalton = halton(n.Din,n.N+1);
        xNext = xhalton(n.N+1,:)./n.dscale+n.dshift;
        [yNext cNext] = functionEval_Spring(xNext);
    else
        xs = varargin{1};
        xNext = xs(n.N+1,:);
        ys = varargin{2};
        yNext = ys(n.N+1,:);
        cs = varargin{3};
        cNext = cs(n.N+1,:);
    end

    % Update the BNC
    n = kbnAddData_Spring(n, xNext, yNext, cNext);
end

end
```

```
function n = kbnAddData_Spring(n,dVal,pVal,cVal)
% Adds a new data point to the structure n, which holds all points and
% all information pertaining to those points.
% Inputs:
%   n = structure variable containing all info about all training
%       points
%   dVal = new design point values
%   cVal = new design point constraint values
%   pVal = new design point performance values
% Outputs:
%   n = updated structure variable with new point information

% Number of new data point(s) to be added (usually only 1)
M = size(dVal,1);

% Add blank new data points to list of points
n.dValues = [n.dValues; zeros(M,n.Din)];
n.cValues = [n.cValues; zeros(M,n.Dcon)];
n.pValues = [n.pValues; zeros(M,n.Dout)];
n.w = [n.w; zeros(M,3)];
n.wd = [n.wd; zeros(M,2)];
n.wp = [n.wp;zeros(M,2)];

% Add M data point(s) to n
for i=1:M
    % Increment counter of # of data points
    n.N = n.N+1;

    % Store the new data point
    n.dValues(n.N,1:n.Din) = dVal(i,1:n.Din);
    n.pValues(n.N,1:n.Dout) = pVal(i,1:n.Dout);
    n.cValues(n.N,1:n.Dout) = cVal(i,1:n.Dcon);

    % Update the standard deviation of the design and performance
    % values
    n.stdD=[std(n.dValues(:,1)) std(n.dValues(:,2))];
    n.stdP=[std(n.pValues(:,1)) std(n.pValues(:,2))];
    n.hd = kbnEvalHd(n);
    n.hp = kbnEvalHp(n);

    % Classify the new design point
    [c cd cp] = classifyDesign(n,pVal(i,:),cVal(i,:));

    % Update previous weights
    n.w(:,c) = n.w(:,c).*n.Nc(c)/(n.Nc(c)+1);
    n.wd(:,cd)=n.wd(:,cd).*n.Ncd(cd)/(n.Ncd(cd)+1);
    n.wp(:,cp)=n.wp(:,cp).*n.Ncp(cp)/(n.Ncp(cp)+1);

    % Increment class counters
    n.Nc(c) = n.Nc(c)+1;
    n.Ncd(cd)=n.Ncd(cd)+1;
```

```
    n.Ncp(cp)=n.Ncp(cp)+1;

    % Add weights for new design
    n.w(n.N,c) = 1/n.Nc(c);
    n.wd(n.N,cd)=1/n.Ncd(cd);
    n.wp(n.N,cp)=1/n.Ncp(cp);
end

end



function kde = kbnEvalDesign(n,xs,varargin)
% This function calculates the kernel density estimates of the the
% design space.
%
% Note: xs must be un-normalized

if ~isempty(varargin)
    s=varargin{1};
else
    s=n.hd;
end

pk = zeros(1,2);
for k=1:2
    for j=1:n.N
        pj=ones(1,2);
        for i=1:n.Din
            xsi=xs(1,i);
            xdi=n.dValues(j,i);
            pji=(1/(2*(s(k,i)^2)*pi())^.5)*exp(-(xsi-xdi)^2 / ...
                (2*s(k,i)^2)); % Normal gaussian distribution
            pj(k)=pj(k)*pji;
        end
        pk(k)=pk(k)+n.wd(j,k)*pj(k);
    end
end
kde=[pk(1) pk(2)];

end



function p = kbnEvalPerf(n,ys,varargin)
% This function calculates the kernel density estimates of the the
% performance space.
%
% Note: ys must be un-normalized
```

```
if ~isempty(varargin)
    s=varargin{1};
else
    s=n.hp;
end

kde=zeros(1,2);
for k=1:2
    for j=1:n.N
        pj=ones(1,2);
        for i=1:n.Dout
            ysi=ys(1,i);
            ydi=n.pValues(j,i);
            pji=(1/(s(k,i)*(2*pi())^.5)) * exp(-0.5*((ysi-ydi) / ...
                (s(k,i)))^2); % Normal gaussian distribution
            pj(k)=pj(k)*pji;
        end
        kde(k)=kde(k)+n.wp(j,k)*pj(k);
    end
end

p=kde;
end
```

```
function h = kbnEvalHd(n)
% Evaluates the standard deviation of the kernels in the performance
% space.

if n.N<=0
    h = ones(1,n.Cd).*n.hdAlpha; % Sets h = [.4 .4]
else
        h = (n.stdD.*n.hdAlpha)./(n.N.^(1/(n.Din))); %=[.4/N^.5]
        h = [h;h];
end

end
```

```
function h = kbnEvalHp(n,varargin)
% Evaluates the standard deviation of the kernels in the performance
% space.

if n.N<=0
    h = ones(1,n.Cp).*n.hpAlpha;
else
    h1=(n.stdP.*n.hpAlpha)./(n.Ncp(1)^(1/n.Dout));
    h2=(n.stdP.*n.hpAlpha)./(n.Ncp(2)^(1/n.Dout));
```

```
    h=[h1;h2];
end

end



function p_thresh = kbnPerfUnattP(n,varargin)

if isempty(varargin)
    type = 'stdAll';
else
    type = varargin{1};
end

switch type
    case 'stdAll'
        p_thresh = 1;
        h = n.stdP;
        for i = 1:n.Dout
            p_thresh = p_thresh*((1/(sqrt(2*pi)*h(i)))*...
                exp(-n.hpBeta/2));
        end
    case 'stdSat'
        ysat=[];
        for i = 1:n.N
            if n.wp(i,1)>0 && n.wd(i,1)>0
                ysat = [ysat;n.pValues(i,:)];
            end
        end
        p_thresh = 1;
        h = std(ysat);
        for i = 1:n.Dout
            p_thresh = p_thresh*((1/(sqrt(2*pi)*h(i)))* ...
                exp(-n.hpBeta/2));
        end
    case 'kernelS'
        p_thresh = 1;
        h = n.hp(1,:);
        for i = 1:n.Dout
            p_thresh = p_thresh*(((1/(sqrt(2*pi)*h(i)))*...
                exp(-n.hpBeta/2))/n.Ncp(1));
        end
    case 'kernelL'
        p_thresh = 1;
        h = n.hp(1,:);
        for i = 1:n.Dout
            p_thresh = p_thresh*((1/(sqrt(2*pi)*h(i)))*...
                exp(-n.hpBeta/2));
        end
    otherwise
        fprintf('%','Error in calculating unattainable pthresh');
```

```
end



function figHandle = plotDSpace(n,divs,sf,varargin)
% Plot the Design Space

if nargin == 0
    load('nDemo100.mat');
    n = nDemo100;
    divs = [40 40];
    sf = [1 1];
end

% Define Preset Plotting Parameters
presetParams.figNum=0;
presetParams.LegendLoc = 'best';
presetParams.axes = [n.dbnd(1,1) n.dbnd(2,1) n.dbnd(1,2) n.dbnd(2,2)];
inputParams = varargin;
plotParams = getPlotParams(presetParams,inputParams);

figHandle=figure(plotParams.figNum);
clf

% Plot the data points
Pc1 = (n.Ncd(1)+1)/(n.N+2);
Pc2 = (n.Ncd(2)+1)/(n.N+2);
if plotParams.DataPoints
    for j=1:n.N
        if strcmp(plotParams.ClassifyData,'actual')
            if n.w(j,1)>0
                plot(n.dValues(j,1),n.dValues(j,2),...
                    'bs','MarkerFaceColor',...
                    'b','MarkerSize',8); hold on;
            elseif n.w(j,2)>0
                plot(n.dValues(j,1),n.dValues(j,2),...
                    'co','MarkerFaceColor',...
                    'c','MarkerSize',8); hold on;
            else
                plot(n.dValues(j,1),n.dValues(j,2),...
                    'r<','MarkerFaceColor',...
                    'r','MarkerSize',8); hold on;
            end
        elseif strcmp(plotParams.ClassifyData,'classifier')
            pTemp = kbnEval_Spring(n,n.d(j,1:n.Din));
            pDiff = (Pc1*pTemp(1)-Pc2*pTemp(2));
            if pDiff>0
                plot(n.d(j,1),n.d(j,2),'b<','MarkerFaceColor','b',...
                    'MarkerSize',8); hold on;
            else
                plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r',...
                    'MarkerSize',8); hold on;
```

```
            end
        else
            display('Invalid Data Point Classification Parameter');
        end
    end
end

% Plot the surfaces
if plotParams.ClassSurf || plotParams.ClassBoundary
    step = [plotParams.axes(2)-plotParams.axes(1) ...
        plotParams.axes(4)-plotParams.axes(3)]./divs;
    x1 = plotParams.axes(1):step(1):plotParams.axes(2);
    x1Count = divs(1)+1;
    x2 = plotParams.axes(3):step(2):plotParams.axes(4);
    x2Count = divs(2)+1;
    x = [x1' x2'];
    kde = zeros(x2Count,x1Count,2);
    dkde = zeros(x2Count,x1Count);
    for i=1:x1Count
        for j=1:x2Count
            kde(j,i,:) = kbnEvalDesign(n,[x(i,1) x(j,2)]);
            dkde(j,i) = (Pc1*sf(1)*kde(j,i,1)-Pc2*sf(2)*kde(j,i,2))/...
                (Pc1*sf(1)*kde(j,i,1)+Pc2*sf(2)*kde(j,i,2));
        end
    end
    if strcmp(plotParams.Orientation,'3D')
        shiftZ = 0;
    else
        shiftZ = max(max(max([Pc1*sf(1)*kde(:,:,1); ...
            Pc2*sf(2)*kde(:,:,2)])))*1.05;
    end
end

if plotParams.ClassSurf
    FaceAlpha = plotParams.ClassSurfFaceAlpha;
    EdgeAlpha = plotParams.ClassSurfEdgeAlpha;
    blue = plotParams.Color.blue;
    red = plotParams.Color.red;
    surface(x(:,1),x(:,2),Pc1*kde(:,:,1)-shiftZ,...
       'FaceAlpha',FaceAlpha,'EdgeColor',...
        blue,'FaceColor',blue,'EdgeAlpha',EdgeAlpha); hold on;
    surface(x(:,1),x(:,2),Pc2*kde(:,:,2)-shiftZ,'FaceAlpha',...
        FaceAlpha,'EdgeColor',red,'FaceColor',red,'EdgeAlpha',...
        EdgeAlpha); hold on;
end

% Plot the decision boundary
if plotParams.ClassBoundary
    contour(x(:,1),x(:,2),dkde,[0 0],'k--','LineWidth',2); hold on;
end

% Set plot parameters
axis(plotParams.axes);
```

```
xlabel('X_1','FontSize',12)
ylabel('X_2','FontSize',12)
legend('Satisfactory','Feasible','Infeasible','Location',...
    plotParams.LegendLoc);

end

% ------------------- Imbeded Function --------------------------------
function plotParams = getPlotParams(presetParams,varargin)
% Define the parameters of the plotPSpace() function of variable input

plotParams = initPlotParams();

% Set the preset params in code above
fields = fieldnames(presetParams);
for i = 1:numel(fields)
    plotParams.(fields{i}) = presetParams.(fields{i});
end

% Set the params inputted into master plotPSpace(varargin)
inputParams = varargin{1};
if ~isempty(inputParams)
    if max(strcmp(inputParams(:),'Orientation'))>0
        ind = find(strcmp(inputParams(:),'Orientation')>0)+1;
        plotParams = paramPreset(plotParams,inputParams(ind));
    end
    for i = 1:2:length(varargin)-1
        if isfield(plotParams,varargin{i})
            plotParams.(varargin{i}) = varargin{i+1};
        else
            fprintf('Incorrect Plot Parameter: %s.\n',varargin{i});
        end
    end
end

% Set figure number to new figure unless specified
if plotParams.figNum < 1
    ag = findobj;
    nf = max(ag(find(ag==fix(ag))));
    plotParams.figNum = nf+1;
end

% Change on/off strings to boolean variables
fields = fieldnames(plotParams);
for i = 1:numel(fields)
    if isa(plotParams.(fields{i}),'char')
        if strcmp(plotParams.(fields{i}),'on')
            plotParams.(fields{i}) = true;
        elseif strcmp(plotParams.(fields{i}),'off')
            plotParams.(fields{i}) = false;
        end
    end
```

```
end

% Set Color Properties
if strcmp(plotParams.Orientation,'3D')
    plotParams.Color.blue = [0 0 1];
    plotParams.Color.red = [1 0 0];
    plotParams.Color.green = [0 1 0];
    plotParams.Color.grey = [.8 .8 .8];
elseif strcmp(plotParams.Orientation,'2D')
    plotParams.Color.blue = [.49 .682 1];
    plotParams.Color.red = [1 .565 .576];
    plotParams.Color.green = [.808 1 .690];
    plotParams.Color.grey = [.8 .8 .8];
end

end

function p = initPlotParams()
% Set the default plotting parameters
    p.DataPoints='on';
    p.ClassifyData='actual';
    p.ClassSurf='on';
    p.ClassSurfEdgeAlpha=0;
    p.ClassSurfFaceAlpha=1;
    p.ClassBoundary='on';
    p.Targets='off';
    p.Orientation='2D';
    p.figNum=0;
    p.interval='off';
end

function p = paramPreset(p,str)
    str = str{1};
    if strcmp(str,'3D')
        p.DataPoints='on';
        p.ClassifyData='actual';
        p.ClassSurf='on';
        p.ClassSurfEdgeAlpha=1;
        p.ClassSurfFaceAlpha=.1;
        p.ClassBoundary='on';

        p.Targets='off';
    elseif strcmp(str,'2D')
        p.DataPoints='on';
        p.ClassifyData='actual';
        p.ClassSurf='on';
        p.ClassSurfEdgeAlpha=0;
        p.ClassSurfFaceAlpha=1;
        p.ClassBoundary='on';
        p.Orientation='2D';
    end
end
```

```
function figHandle = plotPSpace(n,divs,sf,varargin)
% Plot the performance space. Input 'ZoomSatisfactory' to limit to
% satisfacotry performance space

if nargin == 0
    load('nDemo100.mat');
    n = nDemo100;
    divs = [40 40];
    sf = [1 1];
end

% Define Preset Plotting Parameters
inputParams = varargin;
if max(strcmp(inputParams(:),'ZoomSatisfactory'))>0
    presetParams.axes = [n.pbnd(1,1) n.ptarg(1) n.pbnd(1,2) ...
        n.ptarg(2)];
    inputParams = deleteParam(inputParams,'ZoomSatisfactory');
else
    presetParams.axes = [n.pbnd(1,1) n.pbnd(2,1) n.pbnd(1,2)...
        n.pbnd(2,2)];
end
presetParams.figNum = 0; % Setting to 0 will create a new figure
presetParams.LegendLoc = 'best';
plotParams = getPlotParams(presetParams,inputParams);

figHandle=figure(plotParams.figNum);
clf

% Plot the data points
if plotParams.DataPoints
    if plotParams.satPoints
        for j=1:n.N
            if n.w(j,1)>0
                plot(n.pValues(j,1),n.pValues(j,2),...
                    'bs','MarkerFaceColor','b',...
                    'MarkerSize',6); hold on;
            elseif n.w(j,2)>0
                plot(n.pValues(j,1),n.pValues(j,2),...
                    'co','MarkerFaceColor','c',...
                    'MarkerSize',6); hold on;
            else
                plot(n.pValues(j,1),n.pValues(j,2),...
                    'r<','MarkerFaceColor','r',...
                    'MarkerSize',6); hold on;
            end
        end
    else
        for j=1:n.N
            if n.w(j,1)>0 || n.w(j,2)>0
                plot(n.pValues(j,1),n.pValues(j,2),...
```

```
                    'bs','MarkerFaceColor','b',...
                    'MarkerSize',6); hold on;
            else
                plot(n.pValues(j,1),n.pValues(j,2),...
                    'r<','MarkerFaceColor','r',...
                    'MarkerSize',6); hold on;
            end
        end
    end
end

% Plot Performance Targets
if plotParams.Targets
    plot([n.ptarg(1) n.ptarg(1)],[n.pbnd(1,2) n.ptarg(2)],'b--',...
        'LineWidth',2);hold on;
    plot([n.pbnd(1,1) n.ptarg(1)],[n.ptarg(2) n.ptarg(2)],'b--',...
        'LineWidth',2);hold on;
end

% Calculate the Class and Unattainable Surface Data
if plotParams.ClassSurf || plotParams.ClassBoundary ||
plotParams.UnattainableSurf || plotParams.UnattainableBoundary
    step = [plotParams.axes(2)-plotParams.axes(1) ...
        plotParams.axes(4)-plotParams.axes(3)]./divs;
    y1 = plotParams.axes(1):step(1):plotParams.axes(2);
    y1Count = divs(1)+1;
    y2 = plotParams.axes(3):step(2):plotParams.axes(4);
    y2Count = divs(2)+1;
    y = [y1' y2'];
    n.Cp=2;
    c=1:n.Cp;
    kde = zeros(y2Count,y1Count,n.Cp);
    dkde = zeros(y2Count,y1Count);

    dkde_Unattainable = zeros(y1Count,y1Count);
    kde_Unattainable = kbnPerfUnattP(n);
    Pc = (n.Ncp(c)+1)/(n.N+2);
    for i=1:y1Count
        for j=1:y2Count
            kde(j,i,c) = kbnEvalPerf(n,[y(i,1) y(j,2)]);
            if n.Cp > 1
                dkde(j,i) = (Pc(1)*sf(1)*kde(j,i,1)-...
                    Pc(2)*sf(2)*kde(j,i,2))/...
                    (Pc(1)*sf(1)*kde(j,i,1)+Pc(2)*sf(2)*kde(j,i,2));
                if Pc(1)*sf(1)*kde(j,i,1) > Pc(2)*sf(2)*kde(j,i,2)
                    dkde_Unattainable(j,i) = Pc(1)*sf(1)*...
                        kde(j,i,1) -kde_Unattainable;
                else
                    dkde_Unattainable(j,i) = Pc(2)*sf(2)*...
                        kde(j,i,2) - kde_Unattainable;
                end
            else
                dkde_Unattainable(j,i) = Pc*sf(1)*kde(j,i) -...
```

```
                    kde_Unattainable;
            end
        end
    end

    if strcmp(plotParams.Orientation,'3D')
        shiftZ = 0;
    else
        if n.Cp > 1
            shiftZ = max(max(max([Pc(1)*sf(1)*kde(:,:,1); ...
                Pc(2)*sf(2)*kde(:,:,2)])))*1.05;
        else
            shiftZ = max(max(Pc*sf(1)*kde(:,:,1)));
        end
    end
end

if plotParams.ClassSurf
    FaceAlpha = plotParams.ClassSurfFaceAlpha;
    EdgeAlpha = plotParams.ClassSurfEdgeAlpha;
    blue = plotParams.Color.blue;
    red = plotParams.Color.red;
    green = plotParams.Color.green;
    if n.Cp > 1
        surface(y(:,1),y(:,2),Pc(1)*kde(:,:,1)-shiftZ,'FaceAlpha',...
            FaceAlpha,'EdgeColor',blue,'FaceColor',blue,'EdgeAlpha',...
            EdgeAlpha); hold on;
        surface(y(:,1),y(:,2),Pc(2)*kde(:,:,2)-shiftZ,'FaceAlpha',...
            FaceAlpha,'EdgeColor',red,'FaceColor',red,'EdgeAlpha',...
            EdgeAlpha); hold on;
    else
        surface(y(:,1),y(:,2),Pc(1)*kde(:,:,1)-shiftZ,'FaceAlpha',...
            FaceAlpha,'EdgeColor',green,'FaceColor',green,...
           'EdgeAlpha',EdgeAlpha); hold on;
    end
end

% Plot the feasible/infeasible decision boundary
if plotParams.ClassBoundary
    contour(y(:,1),y(:,2),dkde,[0 0],'k--','LineWidth',2); hold on;
end

% Plot the unachievable space
if plotParams.UnattainableSurf
    FaceAlpha = plotParams.UnattainableSurfFaceAlpha;
    EdgeAlpha = plotParams.UnattainableSurfEdgeAlpha;
    grey = plotParams.Color.grey;
    divsUnatt = [40 40];
    xUnach=linspace(plotParams.axes(1), plotParams.axes(2),...
        divsUnatt(1)+1);
    yUnach=linspace(plotParams.axes(3), plotParams.axes(4),...
        divsUnatt(2)+1);
    zUnach=ones(divsUnatt(1)+1,divsUnatt(2)+1).*kde_Unattainable;
```

```
    surface(xUnach,yUnach,zUnach-shiftZ,'FaceAlpha',FaceAlpha,...
        'EdgeAlpha',EdgeAlpha,'FaceColor',grey); hold on;
end

if plotParams.UnattainableBoundary
    contour(y(:,1),y(:,2),dkde_Unattainable,[0 0],'k-',...
        'LineWidth',2); hold on;
end

% Set plot parameters
axis(plotParams.axes);
xlabel('Y_1','FontSize',12)
ylabel('Y_2','FontSize',12)
if plotParams.Legend
    legend('Satisfactory','Feasible','Infeasible',...
        'Location',plotParams.LegendLoc);
end
view(plotParams.view);

end
% ------------------- Imbeded Function --------------------------------
function plotParams = getPlotParams(presetParams,varargin)
% Define the parameters of the plotPSpace() function of variable input

plotParams = initPlotParams();

% Set the preset params in code above
fields = fieldnames(presetParams);
for i = 1:numel(fields)
    plotParams.(fields{i}) = presetParams.(fields{i});
end

% Set the params inputted into master plotPSpace(varargin)
inputParams = varargin{1};
if ~isempty(inputParams)
    if max(strcmp(inputParams(:),'Orientation'))>0
        ind = find(strcmp(inputParams(:),'Orientation')>0)+1;
        plotParams = paramPreset(plotParams,inputParams(ind));
    end
    for i = 1:2:length(inputParams)-1
        if isfield(plotParams,inputParams{i})
            plotParams.(inputParams{i}) = inputParams{i+1};
        else
            fprintf('Incorrect Plot Parameter: %s.\n',varargin{i});
        end
    end
end

% Set figure number to new figure unless specified
if plotParams.figNum < 1
    ag = findobj;
    nf = max(ag(find(ag==fix(ag))));
```

```
    plotParams.figNum = nf+1;
end

% Change on/off strings to boolean variables
fields = fieldnames(plotParams);
for i = 1:numel(fields)
    if isa(plotParams.(fields{i}),'char')
        if strcmp(plotParams.(fields{i}),'on')
            plotParams.(fields{i}) = true;
        elseif strcmp(plotParams.(fields{i}),'off')
            plotParams.(fields{i}) = false;
        end
    end
end

% Set Color Properties
if strcmp(plotParams.Orientation,'3D')
    plotParams.Color.blue = [0 0 1];
    plotParams.Color.red = [1 0 0];
    plotParams.Color.green = [0 1 0];
    plotParams.Color.grey = [.4 .4 .4];
elseif strcmp(plotParams.Orientation,'2D')
    plotParams.Color.blue = [.49 .682 1];
    plotParams.Color.red = [1 .565 .576];
    plotParams.Color.green = [.808 1 .690];
    plotParams.Color.grey = [.8 .8 .8];
end

end

function p = initPlotParams()
% Set the default plotting parameters
    p.DataPoints='on';
    p.ClassSurf='on';
    p.ClassSurfEdgeAlpha=0;
    p.ClassSurfFaceAlpha=1;
    p.ClassBoundary='on';
    p.UnattainableSurf='on';
    p.UnattainableSurfEdgeAlpha=.2;
    p.UnattainableSurfFaceAlpha=1;
    p.UnattainableBoundary='on';
    p.Targets='off';
    p.Orientation='2D';
    p.view=3;
    p.figNum=0;
    p.Legend='on';
    p.satPoints='on';
end

function p = paramPreset(p,str)
    str = str{1};
    if strcmp(str,'3D')
```

```
        p.DataPoints='on';
        p.ClassSurf='on';
        p.ClassSurfEdgeAlpha=1;
        p.ClassSurfFaceAlpha=.1;
        p.ClassBoundary='on';
        p.UnattainableSurf='on';
        p.UnattainableSurfEdgeAlpha=.1;
        p.UnattainableSurfFaceAlpha=.2;
        p.UnattainableBoundary='on';
        p.Targets='off';
        p.view=3;
    elseif strcmp(str,'2D')
        p.DataPoints='on';
        p.ClassSurf='on';
        p.ClassSurfEdgeAlpha=0;
        p.ClassSurfFaceAlpha=1;
        p.ClassBoundary='on';
        p.UnattainableSurf='on';
        p.UnattainableSurfEdgeAlpha=.2;
        p.UnattainableSurfFaceAlpha=1;
        p.UnattainableBoundary='on';
        p.Targets='off';
        p.Orientation='2D';
        p.view=2;
    end

end

function p = deleteParam(p,param2Delete)
    ind = find(strcmp(p(:),param2Delete)>0);
    if ind == 1
        if numel(p) == 1
            p = {};
        else
            p = p(2:end);
        end
    else
        if numel(p) == ind
            p = p(1:end-1);
        else
            ptemp(1:ind-1) = p(1:ind-1);
            ptemp(ind+1:end) = p(ind+1:end);
            p=ptemp;
        end
    end

end




function MC = kbnMonteCarlo(n,spaceEval,classEval,N,sf)
```

```
% Perform Monte Carlo volume integration of design/performance space
%
% Inputs:
%   spaceEval = 'd' - design space
%             = 'p' - performance space
%   classEvak = '1' - satisfactory space (feasible and meets
%                     constraints)
%             = '2' - feasible, but not meet constraints
%             = '3' - Infeasible
%             = [1,3] - Infeasible AND satisfactory (put multiple
%                       classes
%                       in a sorted array)
%   N = Number of sample evaluations for the Monte Carlo
%   sf = Scaling Factor for classes

pDesiredSpace = 1;
pAllSpace = ~pDesiredSpace;

MC = struct('sum',0);
MC.tally = [];

MC.volume = 0;
MC.volumeSummary = [];
MC.variance = [];
MC.varianceSummary = [];


% Get constant kbn variable values
numKbnClasses = 2; % For 2 classes
c = 1:numKbnClasses;
switch spaceEval
    case 'd'
        Pc = (n.Ncd(c)+1)./(n.N+2);
        scale = n.dscale;
        shift = n.dshift;
    case 'p'
        if pDesiredSpace
            pbnd = [n.pbnd(1,:); n.ptarg];
            scale = 1./(pbnd(2,:)-pbnd(1,:));
            shift = pbnd(1,:);
        end
        if pAllSpace
            scale = n.pscale;
            shift = n.pshift;
        end

        attainableProbBnd = kbnPerfUnattP(n);
        Pc = (n.Ncp(c)+1)./(n.N+2);
    otherwise
        fprintf('Invalid spaceEval in kbnMonteCarlo()');
end
```

```
numVars = 2; % Number of variables / Number of dimensions of space
testVars = halton(numVars,N+1000);
testVars = testVars(1001:end,:);

for i = 1:N
    switch spaceEval
        case 'd'
            pTemp = kbnEvalDesign(n,testVars(i,:)./scale + shift);
        case 'p'
            pTemp = kbnEvalPerf(n,testVars(i,:)./scale + shift);
    end

    pDiff = (sf(1)*Pc(1)*pTemp(1)-sf(2)*Pc(2)*pTemp(2));

    switch spaceEval
        case 'd'
            if pDiff >= 0 && max(classEval==1)==1
                MC = updateInsideMC(MC);
            elseif pDiff < 0 && (max(classEval==2)==1 ||
max(classEval==3)==1)
                MC = updateInsideMC(MC);
            else
                MC = updateOutsideMC(MC);
            end
        case 'p'
            if pDiff >= 0 && max(pTemp)*Pc(1)*sf(1)>=attainableProbBnd
                if isSatisfactory(n,testVars(i,:)) &&
max(classEval==1)==1
                    MC = updateInsideMC(MC);
                elseif ~isSatisfactory(n,testVars(i,:)) &&
max(classEval==1)==1
                    MC = updateInsideMC(MC);
                end
            elseif pDiff <= 0 &&
max(pTemp)*Pc(2)*sf(2)>=attainableProbBnd && max(classEval==3)==1
                MC = updateInsideMC(MC);
            else
                MC = updateOutsideMC(MC);
            end
    end
end

end

function newMC = updateInsideMC(MC)
% Update MC for a new sample inside
    newMC = MC;

    newMC.sum = newMC.sum + 1;
    if isfield(MC,'tally')
        newMC.tally = [newMC.tally; 1];
    end
```

```
    if isfield(MC,'volume')
        newMC.volume = newMC.sum/length(MC.tally);
    end
    if isfield(MC,'volumeSummary')
        newMC.volumeSummary = [MC.volumeSummary; newMC.volume];
    end
    if isfield(MC,'variance')
        newMC.variance = (newMC.volume*(1 - newMC.volume))/...
            (length(MC.tally)-1);
    end
    if isfield(MC,'varianceSummary')
        newMC.varianceSummary = [newMC.varianceSummary; ...
            newMC.variance];
    end
end

function newMC = updateOutsideMC(MC)
% Update MC for a new sample outside
    newMC = MC;

    if isfield(MC,'tally')
        newMC.tally = [newMC.tally; 0];
    end
    if isfield(MC,'volume')
        newMC.volume = newMC.sum/length(MC.tally);
    end
    if isfield(MC,'volumeSummary')
        newMC.volumeSummary = [newMC.volumeSummary; newMC.volume];
    end
    if isfield(MC,'variance')
        newMC.variance = (newMC.volume*(1 - newMC.volume))/...
            (length(MC.tally)-1);
    end
    if isfield(MC,'varianceSummary')
        newMC.varianceSummary = [newMC.varianceSummary; ...
            newMC.variance];
    end
end

function YES = isSatisfactory(n,yi)
    YES = true;

    for i = size(yi,2);
        if yi(i) > (n.ptarg(i)-n.pshift(i))*n.pscale(i)
            YES = false;
        end
    end
end



function xs = halton(D,N)
```

```
% Finds a Halton sequence of data points in the D dimensional design
% space for N data points.  All points are between 0 and 1 for all
% variables.

xs = zeros(N,D);  % Initialize xs to hold N data points
primes10=primes(30); % Returns the first 10 prime numbers
ps = primes10(1:D);  % ps is the first D prime numbers

for j = 1:N % Fill out xs
   x = zeros(1,D);
   base = ps;
   index = j*ones(1,D);
   while any(index)
       digit = mod(index,ps); % This line is a problem for D>2
       x = x + digit./base;
       index = (index-digit)./ps;
       base = base.*ps;
   end
   xs(j,:) = x;
end

end
```

Bibliography

Antonsson, E. K., & Otto, K. N. (1995). Imprecision in Engineering Design. Journal of Mechanical Design, 117(B), 25–32. Retrieved from http://design.caltech.edu/Research/Imprecise/Papers/94g.pdf

Backlund, P. B. (2012). A Classifier-Guided Sampling Method for Early-Stage Design of Shipboard Energy Systems. University of Texas at Austin.

Becz, S., Se, L., Street, D., Ma, U., Pinto, A., Zeidner, L. E., Khire, R., et al. (2010). Design System for Managing Complexity in Aerospace Systems. AIAA ATIO/ISSMO Conference (pp. 1–7).

Bosman, P., & Thierens, D. (2000). IDEAs Based on the Normal Kernels Probability Density Function. Utrecht University Technical Report, (UU-CS-2000-11). Retrieved from http://igitur-archive.library.uu.nl/math/2001-0220-120742/UUindex.html

Chang, T. S., Ward, A. C., Lee, J., & Jacox, E. H. (1994). Conceptual Robustness in Simultaneous Engineering: An Extension of Taguchi's Parameter Design. Research in Engineering Design, 6(4), 211–222. Retrieved from http://www.springerlink.com/index/R481L6U484407265.pdf

Chen, R., & Ward, A. C. (1995). The RANGE family of propagation operations for Intervals on simultaneous linear equations. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 9, 183–196.

Chen, W., Simpson, T. W., Allen, J. K., & Mistree, F. (1996). Using Design Capability Indices to satisfy Ranged Set of Design Requirements. In ASME Design

Engineering Technical Conferences and Computers in Engineering Conference (pp. 1–11). Retrieved from http://www.srl.gatech.edu/publications/1996/fm.ja.chen.confpro.DAC-1090.1996.pdf

Chen, W., & Yuan, C. (1998). A Probabilistic-Based Design Model for Achieving Flexibility in Design. ASME Journal of Mechanical Design, (312), 1–33.

Chen, W. (1999). A Robust Design Approach for Achieving Flexibility in Multidisciplinary Design. AIAA Journal, 37(8), 982–989.

Chen, W., Yin, X., Lee, S., & Liu, W. K. (2010). A Multiscale Design Methodology for Hierarchical Systems With Random Field Uncertainty. Journal of Mechanical Design, 132(4), 041006 (11 pages). doi:10.1115/1.4001210

Choi, H., McDowell, D. L., Allen, J. K., Rosen, D., & Mistree, F. (2008). An inductive design exploration method for robust multiscale materials design. Journal of Mechanical Design, 130, 031402.

Clark, K. B., & Fujimoto, T. (1991). Product Development Performance: Strategy, Organization, and Management in the World Auto Industry. Boston, MA: Harvard Business Press.

Clevenger, C., & Haymaker, J. (2011). Metrics to Assess Design Guidance. Design Studies, 32(5), 431–456. Retrieved from http://www.sciencedirect.com/science/article/pii/S0142694X11000135

Davis, E. (1987). Constraint Propagation with Interval Labels. Artificial Intelligence, 32(3), 281–331. doi:10.1016/0004-3702(87)90091-9

Devendorf, E., & Lewis, K. (2008). Planning on Mistakes: An Approach to Incorporate Error Checking into the Design Process. ASME International Design Engineering Technical Conferences (pp. 1–12). Brooklyn, New York.

Dudda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern Classification (2nd ed.). New York: John Wiley & Sons.

Eggert, R. J., & Mayne, R. W. (1993). Probabilistic Optimal Design Using Successive Surrogate Probability Density Functions. Journal of Mechanical Design, 115(September), 385–391.

Evans, M., & Swartz, T. (2000). Approximating Integrals via Monte Carlo and Deterministic Methods. Oxford: Oxford University Press. Retrieved from http://books.google.com/books?id=GMHKfx84L4MC&pg=PA276&dq=monte+carlo+berger&hl=en&sa=X&ei=AwDiUfDFMcXLyQG5noGQBg&ved=0CD0Q6wEwAg#v=onepage&q=monte carlo berger&f=false

Ferguson, S., & Lewis, K. (2006). Effective Development of Reconfigurable Systems Using Linear State-Feedback Control. AIAA Journal, 44(4), 868–878. doi:10.2514/1.17147

Ferguson, S., Siddiqi, A., Lewis, K., & de Weck, O. (2007). Flexible and reconfigurable systems: Nomenclature and review. 2007 ASME DETC and CIE Conferences, 1–15. Retrieved from http://people.engr.ncsu.edu/smfergu2/sites/default/files/papers/DETC2007-35745.pdf

Finch, W., & Ward, A. (1997). A Set-Based System for Eliminating Infeasible Designs in Engineering Problems Dominated by Uncertainty. ASME Design Engineering Technical Conferences, 1–12. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.156&rep=rep1&type=pdf

Fishman, G. S. (1996). Monte Carlo: Concepts, Algorithms, and Applications. New York: Springer-Verlag New York, Inc.

Freeman, G., & Halton, J. (1951). Note on an Exact Treatment of Contingency, Goodness of Fit and Other Problems of Significance. Biometrika, 38(1/2), 141–149. Retrieved from http://www.jstor.org/stable/10.2307/2332323

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian Network Classifiers. Machine Learning, 29, 131–163.

Fukunaga, K. (1990). Introduction to Statistical Pattern Recognition (2nd ed.). San Diego, CA: Academic Press.

Gertler, J. (2012). F-35 Joint Strike Fighter (JSF) Program. Washington, D.C.

Gupta, Y. P., & Goyal, S. (1990). Flexibility of Manufacturing Systems: Concepts and Measurements. European Journal of Operational Research, 43(2), 119–135.

Gurnani, A., & Lewis, K. L. (2008). Collaborative, Decentralized Engineering Design at the Edge of Rationality. Journal of Mechanical Design, 130(12), 121101. doi:10.1115/1.2988479

Haberman, M. R., Klatt, T. D., Wilson, P. S., & Seepersad, C. C., (2012). Negative Stiffness Metamaterials and Periodic Composites. J. Acoust. Soc. Am., 131(4).

Haberman, M. R., Berthelot, Y. H., & Cherkaoui, M. (2006). Micromechanical Modeling of Particulate Composites for Damping of Acoustic Waves. Journal of Engineering Materials and Technology, 128(3), 320. doi:10.1115/1.2204943

Haftka, R. T. (1985). Simultaneous Analysis and Design. AIAA Journal, 23(7), 1099–1103. doi:10.2514/3.9043

Hammersley, J. (1960). Monte Carlo Methods for Solving Multivariable Problems. Annals of the New York Academy of Sciences, 86, 844–874. Retrieved from http://onlinelibrary.wiley.com/doi/10.1111/j.1749-6632.1960.tb42846.x/abstract

Hoffmann, R., & Tresp, V. (1996). Discovering Structure in Continuous Variables Using Bayesian Networks. Advances in Neural Information Processing Systems, 500–506. Retrieved from http://www.tresp.org/papers/bayes.pdf

Ilkka, A. (1985). Criteria Changes Across Product Development Stages. Industrial Marketing Management, 14, 171–178.

Jiao, J., & Tseng, M. M. (2004). Customizability Analysis in Design for Mass Customization. Computer-Aided Design, 36(8), 745–757. doi:10.1016/j.cad.2003.09.012

John, G., & Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, 338–345. Retrieved from http://dl.acm.org/citation.cfm?id=2074196

Jones, M., Marron, J., & Sheather, S. (1996). A brief survey of bandwidth selection for density estimation. Journal of the American Statistical Association, 91(433), 401–

407. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/01621459.1996.10476701

Juvinall, R. C., & Marshak, K. M. (2000). Fundamentals of Machine Component Design (3rd ed.). New York: John Wiley & Sons, Inc.

Kalsi, M., Hacker, K., & Lewis, K. (1999). A Comprehensive Robust Design Approach for Decision Trade-offs in Complex Systems Design. ASME Design Engineering Technical Conferences and Computers in Engineering Conference, 1–12. Retrieved from http://link.aip.org/link/?JMDEDB/123/1/1

Katkovnik, V., & Shmulevich, I. (2002). Kernel density estimation with adaptive varying window size. Pattern Recognition Letters, 23(14), 1641–1648. doi:10.1016/S0167-8655(02)00127-7

Kim, H. M., Michelena, N. F., Papalambros, P. Y., & Jiang, T. (2003). Target Cascading in Optimal System Design. Journal of Mechanical Design, 125(3), 474. doi:10.1115/1.1582501

Kim, J., & Wilecon, D. (2009). An Empirical Investigation of Complexity and its Management in New Product Development. Technology Analysis & Strategic Management, 21(4), 547–564.

Klatt, T., & Haberman, M. R. (2013). A nonlinear negative stiffness metamaterial unit cell and small-on-large multiscale material model. Journal of Applied Physics, 114(3), 033503-033503.

Klein, M., Sayama, H., Faratin, P., & Bar-Yam, Y. (2003). The Dynamics of Collaborative Design: Insights from Complex Systems and Negotiation Research. Concurrent Engineering, 11(3), 201–209. doi:10.1177/106329303038029

Koutsawa, Y., Haberman, M. R., Daya, E. M., & Cherkaoui, M. (2008). Multiscale Design of a Rectangular Sandwich Plate with Viscoelastic Core and Supported at Extents by Viscoelastic Materials. International Journal of Mechanics and Materials in Design, 5(1), 29–44. doi:10.1007/s10999-008-9084-0

Kovar, D., King, B. H., Trice, R. W., & Halloran, J. W. (1997). Fibrous Monolithic Ceramics. Journal of the American Ceramic Society, 80(10), 2471–2487.

Kroo, I., Altus, S., Braun, R., Gage, P., & Sobieski, I. (1994). Multidisciplinary Optimization Methods for Aircraft Preliminary Design. 5th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, (AIAA Paper 94-4325), 697–707. Retrieved from http://aero.stanford.edu/reports/mdo94.html

Kursweil, R. (2008). The Coming Merging of Mind and Machine. Scientific American, 18, 20–25.

Lakes, R. (2001). Extreme Damping in Composite Materials with a Negative Stiffness Phase. Physical Review Letters, 86(13), 2897–2900. doi:10.1103/PhysRevLett.86.2897

Lakes, R. S., Lee, T., Bersie, A., & Wang, Y. C. (2001). Extreme Damping in Composite Materials with Negative-Stiffness Inclusions. Nature, 410(6828), 565–7. doi:10.1038/35069035

Ledl, T. (2004). Kernel density estimation: theory and application in discriminant analysis. Austrian journal of statistics, 33(3), 267-279.

Liu, H., Chen, W., Scott, M. J., & Qureshi, K. (2008). Determination of Ranged Sets of Design Specifications by Incorporating Design-Space Heterogeneity. Engineering Optimization, 40(11), 1011–1029. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/03052150802378558

Mattson, C. A., & Messac, A. (2003). Concept Selection using s-Pareto Frontiers Concept Selection Using s-Pareto Frontiers. AIAA, 41(6), 1190–1198.

Milton, G. W. (2002). The Theory of Composites. New York: Cambridge University Press.

Odegard, G. M. (2004). Constitutive Modeling of Piezoelectric Polymer Composites. Acta Materialia, 52(18), 5315–5330. doi:10.1016/j.actamat.2004.07.037

Olewnik, A., & Lewis, K. (2006). A Decision Support Framework for Flexible System Design. Journal of Engineering Design, 17(1), 75–97. doi:10.1080/09544820500274019

Otto, K., & Antonsson, E. (1993). Extensions to the Taguchi Method of Product Design. Journal of Mechanical Design, 115(1), 5–15. Retrieved from http://design.caltech.edu/Research/Publications/90f.pdf

Pacheco, J. E., Amon, C. H., & Finger, S. (2003). Bayesian Surrogates Applied to Conceptual Stages of the Engineering Design Process. Journal of Mechanical Design, 125(4), 664. doi:10.1115/1.1631580

Panchal, J. H., & Allen, J. (2005). An Interval-Based Focalization Method for Decision-Making in Decentralized, Multi-Functional Design. In ASME Design Engineering Technical Conferences and Computers in Engineering Conference (pp. 1–14). Retrieved from http://www.srl.gatech.edu/publications/2005/ja.cp.fm.panchal.fernandez.confpro.DETC85322.pdf

Panchal, J. H., Fernandez, M. G., Paredis, C. J., Allen, J. K., & Mistree, F. (2007). An Interval-based Constraint Satisfaction (IBCS) Method for Decentralized, Collaborative Multifunctional Design. Concurrent Engineering: Research and Applications, 15(3), 309–323. doi:10.1177/1063293X07083083

Pareto, V. (1971). Manual of Political Economy, (Translated into English by A.S. Schwier. Original Work Published 1906 Manuale di Econòmica Polìttica). Milan, Italy: Macmillan.

Parkinson, A., Sorensen, C., & Pourhassan, N. (1993). A General Approach for Robust Optimal Design. Journal of Mechanical Design, (March), 4–10.

Parzen, E. (1962). On estimation of a probability density function and mode. The annals of mathematical statistics, 225(21), 1065–1076. Retrieved from http://www.jstor.org/stable/10.2307/2237880

Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Pérez, A., Larrañaga, P., & Inza, I. (2009). Bayesian Classifiers Based on Kernel Density Estimation: Flexible Classifier. International Journal of Approximate Reasoning, 50(2), 341–362. doi:10.1016/j.ijar.2008.08.008

Reddy, R., & Mistree, F. (1992). Modeling Uncertainty in Selection Using Exact Interval Arithmetic. Design Theory and Methodology, ASME DE-Vo, 193–201. Retrieved from http://www.srl.gatech.edu/publications/1992/fm.peddy.confpro.ASME.1992.pdf

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. The Annals of Mathematical Statistics, 27(3), 832–837. Retrieved from http://www.jstor.org/stable/10.2307/2237390

Ross, D., Ungar, E., & Kerwin, E. (1959). Damping of Plate Flexural Vibrations by Means of Viscoelastic Laminae. Structural damping, 3, 44–87. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Damping+of+Plate+Flexural+Vibrations+by+Means+of+Viscoelastic+Laminae#0

Saridakis, K. M., & Dentsoras, A. J. (2006). Integration of fuzzy logic, genetic algorithms and neural networks in collaborative parametric design. Advanced Engineering Informatics, 20(4), 379–399. doi:10.1016/j.aei.2006.06.001

Scott, D. W. (1992). Multivariate Density Estimates. New York: John Wiley & Sons.

Shahan, D.W., & Seepersad, C. C. (2010a). Implications of Alternative Multilevel Design Methods for Design Process Management. Concurrent Engineering, (512), 1–35. Retrieved from http://cer.sagepub.com/content/18/1/5.short

Shahan, D.W., & Seepersad, C. C. (2010b). Bayesian Network Classifiers for Set-based Collaborative Design. In ASME Design Engineering Technical Conferences and Computers in Engineering Conference (pp. 1–11). ASME. Retrieved from http://link.aip.org/link/abstract/ASMECP/v2010/i44090/p523/s1

Shahan, D. W., & Seepersad, C. C. (2012). Bayesian Network Classifiers for Set-Based Collaborative Design. Journal of Mechanical Design, 134(7), 071001. doi:10.1115/1.4006323

Shawe-Taylor, J., & Cristianini, N. (2004). Kernel Methods for Pattern Analysis (1st ed.). Cambridge, UK: Cambridge University Press.

Silverman, B. (1986). Density Estimation for Statistics and Data Analysis. London: Chapman and Hall.

Simpson, T., Maier, J., & Mistree, F. (2001). Product Platform Design: Method and Application. Research in engineering Design, 13(1), 2–22. Retrieved from http://www.springerlink.com/index/pdf/10.1007/s001630100002

Simpson, T., Rosen, D., Allen, J., & Mistree, F. (1998). Metrics for Assessing Design Freedom and Information Certainty in the Early Stages of Design. Journal of Mechanical Design, 120(4), 628–635. Retrieved from http://www.srl.gatech.edu/publications/1996/fm.dr.ja.simpson.confpro.DTM1521.1996.pdf

Skowronski, V. J., & Turner, J. U. (1997). Using Monte-Carlo variance reduction in statistical tolerance synthesis. Computer-Aided Design, 29(1), 63–69. doi:10.1016/S0010-4485(96)00050-4

Sobek, D. K., Ward, A. C., & Liker, J. K. (1999). Toyota's Principles of Set-Based Concurrent Engineering. Sloan Management Review, 40(2), 67–84. Retrieved from http://6sigma.mty.itesm.mx/Toyotas.pdf

Sobieszczanski-Sobieski, J, & Kodiyalam, S. (1999). BLISS/S: A New Method for Two-Level Structural Optimization. AIAA/ASME/ASCE/AHS/ASC Structural Dynamics, and Materials Conference (Vol. 21, pp. 1–13). St. Louis, MO. Retrieved from http://www.springerlink.com/index/DDPDB7DB7NY71CKN.pdf

Sobieszczanski-Sobieski, Jaroslaw. (1988). Optimization by Decomposition: A Step from Hierarchic to Non-Hierarchic Systems. Second NASA/Air Force Symposium on Recent Advances in Multidisciplinary Analysis and Optimization (p. NASA TM–101494. NASA CP–3031.). Hampton, VA. Retrieved from http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19890015775_1989015775.pdf#page=64

Suh, N. (1990). The Principles of Design. New York: Oxford University Press.

Sweetman, B. (2012). JSF Acquisition Malpractice. Aviation Weekly/Ares Blog. Retrieved October 10, 2012, from http://www.aviationweek.com/Blogs.aspx?plckBlogId=Blog:27ec4a53-dcc8-42d0-bd3a-01329aef79a7&plckController=Blog&plckScript=blogScript&plckElementId=blogDest&plckBlogPage=BlogViewPost&plckPostId=Blog%3A27ec4a53-dcc8-42d0-bd3a-01329aef79a7Post%3A2a8f87e0-ad8d-4b78-97ac-f87851e1e0c0

Taguchi, G., & Cariapa, V. (1993). Taguchi on Robust Technology Development. Journal of Pressure Vessel Technology, 115(August), 336–337. Retrieved from http://link.aip.org/link/?JPVTAS/115/336/1

Thurston, D. L. (1991). A Formal Method for Subjective Design Evaluation with Multiple Attributes. Research in Engineering Design, 3(2), 105–122. doi:10.1007/BF01581343

Toni, A. De, & Tonchia, S. (1998). Manufacturing flexibility: a literature review. International Journal of Production …, (November 2012), 37–41. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/002075498193183

Tsui, K. L., Allen, J. K., Chen, W., & Mistree, F. (1996). A Procedure for Robust Design: Minimizing Variations Caused by Noise Factors and Control Factors. ASME Journal of Mechanical Design, 118, 478–485. Retrieved from http://www.srl.gatech.edu/publications/1996/fm.ja.chen.journ.ASME.vol118.1996.pdf

Voigt, W. (1889). Uber die Beziehung Zurischen den Beiden Elastizitäts Konstanten Isotroperkörper. Wied. Ann, 38, pp. 573–587.

von Neumann, J., & Morgenstern, O. (1947). Theory of games and economic behavior. Princeton University, Princeton (2nd ed.). Princeton, NJ: Princeton University Press. Retrieved from http://library.wur.nl/WebQuery/clc/482840

Wallace, D. R., Jakiela, M. J., & Flowers, W. C. (1996). Design search under probabilistic specifications using genetic algorithms. Computer-Aided Design, 28(5), 405–421.

Ward, A. C., Lozano-Perez, T., & Seering, W. (1990). Extending the Constraint Propagation of Intervals. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 4(1), 47–54. Retrieved from http://journals.cambridge.org/production/action/cjoGetFulltext?fulltextid=4198688

Ward, A. C., Liker, J., Cristiano, J., & Sobek, D. (1995). The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster. Sloan Management …, 36(3), 43–61. Retrieved from http://www.columbia.edu/itc/sociology/watts/g9058/client_edit/ward_et_al.pdf

Wood, K. L., & Antonsson, E. K. (1989). Computations with imprecise parameters in engineering design: background and theory. ASME Journal of Mechanisms, Transmissions, and Automation in Design, 111(4), 616–625. Retrieved from http://www.design.caltech.edu/Research/Publications/88a.pdf

Wood, W. H., & Agogino, A. M. (2005). Decision-Based Conceptual Design: Modeling and Navigating Heterogeneous Design Spaces. Journal of Mechanical Design, 127(1), 2. doi:10.1115/1.1799612

Wujek, B. A., Renaud, J. E., Batill, S. M., & Brockman, J. B. (1996). Concurrent Subspace Optimization Using Design Variable Sharing in a Distributed Computing Environment. Concurrent Engineering, 4(4), 361–377.