Copyright

by

David Williamson Shahan

2010

The Dissertation Committee for David Williamson Shahan certifies that this is the approved version of the following dissertation:

Bayesian Network Classifiers for Set-Based Collaborative Design

Committee:

Carolyn Conner Seepersad, Supervisor

Jason Augenbaugh

Matthew I. Campbell

Richard H. Crawford

Elmira Popova

Kristin L. Wood

Bayesian Network Classifiers for Set-Based Collaborative Design

by

David Williamson Shahan, B.S; M.S.

Dissertation

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

Doctor of Philosophy

The University of Texas at Austin
December 2010

Acknowledgements

I would like to acknowledge Dr. Seepersad for all of the support, guidance and insight she has provided me over the years as my advisor. Dr. Seepersad's energy and enthusiasm for research is contagious, and I feel lucky to have been under her tutelage. I would also like to recognize my committee members for their time, attention, and insights over the years—I have learned a lot from them all. Kaarthic Madhavan should also be recognized as an early collaborator on set-based design whose work has been influential on mine. Steve Embleton, Auhona Hoq, and Jonathan LeSage worked many hours with me to produce the unmanned aerial vehicle simulation that is used extensively in this research. I am thankful for the gracious support of the University of Texas at Austin, the National Science Foundation (NSF grant DMI-0632766), and for the additional funding I received through the David Burton, Jr. Fellowship and the Thrust 2000 Fellowship. I would also like to recognize the importance of my friends and family whose support over the years has been wonderful. In particular, the love and support of my wife Susan has been truly a blessing.

Bayesian Network Classifiers for Set-Based Collaborative Design

David Williamson Shahan, Ph.D.
The University of Texas at Austin, 2010

Supervisor: Carolyn Conner Seepersad

For many products, the design process is a complex system involving the interaction of many distributed design activities that need to be carefully coordinated. This research develops a new tool, called a Bayesian network classifier, to improve one specific aspect of this challenge: quantitatively capturing a consensus of which designs are feasible options for meeting system-wide engineering requirements. Classifiers enable designers to independently develop and share maps of the feasible regions of their design space, enabling set-based collaborative design. The method is set-based in that resources are used to thoroughly understand design tradeoffs before commitment is made to a final design. The method is collaborative because the maps are coordinated between design teams to represent the mutually feasible design space of all stake-holders. The benefits are a more thorough understanding of the system-wide design problem across team boundaries as well as knowledge capture for future re-use, potentially leading to faster product development and higher quality products.

Table of Contents

List of Tables viii

List of Figures ix

Chapter 1: Introduction 1

Chapter 2: The Collaborative Design Research Context 18

Chapter 3. The Kernel-Based Bayesian Network Classifier 31
  3.1 Bayesian Decision Theory 32
  3.2 Kernel Density Estimation and Bayesian Network Classifiers 34
  3.3 Evaluating, Sampling, and Training 39
  3.4 The KBN Classifier's Settings 42
  3.5 Representation Capability 44
  3.6 Discussion 47

Chapter 4. Classifiers for Collaborative Design: Vertical Coupling 50
  4.1 The UAV Demonstration Problem 50
  4.2 UAV Solution 1: Baseline with Vertical Coupling Only 53
  4.3 UAV Solution 2: Baseline Plus Loss Factors 64
  4.4 Discussion 65

Chapter 5. Adaptively Setting the Standard Deviation 69
  5.1 Background 69
  5.2 The Proposed Method 74
  5.3 UAV Solution 3: Baseline Plus Adaptive Standard Deviations 78
  5.4 Discussion 87

Chapter 6. Incorporating Knowledge of Monotonic Relationships 92
  6.1 Background 94
  6.2 The Proposed Methods 96
  6.3 UAV Solution 4: Baseline Plus Seeded Designs 98
  6.4 UAV Solution 5: Baseline Plus Automatic Classification 101
  6.5 Discussion 104

Chapter 7. Adaptive Sampling 107

- 7.1 KBN Classifiers for Exploratory Sampling 108
- 7.2 Background 113
- 7.3 UAV Solution 6: Exploratory Sampling 119
- 7.4 UAV Solutions 7 and 8: Exploratory and Exploitive Sampling 122
- 7.5 Discussion 131

Chapter 8. Cross-Classification and Collaborative Classification Networks 134

- 8.1 Collaborative Classification Networks 135
- 8.2 Cross-Classification 141
- 8.3 UAV Solution 9: Cross-Classification and Separated Subsystems 145
- 8.4 Discussion 150

Chapter 9. Conclusions 158

- 9.1 Achievements 158
- 9.2 Discussion and Future Work 160
- 9.3 Practical Implications 165

Appendix A 167

Appendix B 178

Appendix C 204

Appendix D 230

Appendix E 246

Appendix F 256

Appendix G 275

Bibliography 282

List of Tables

Table 1.1: Research Requirements 16

Table 3.1: Time Complexity for using the KBN Classifier 41

Table 4.1: UAV Design Parameter Limits...56

Table A.1: Matlab® Functions for using KBN's...167

Table B.1: Matlab® Functions for the UAV simulations.....................................178

Table C.1: Matlab® Functions for UAV Solutions 1 and 2 204

Table D.1: Matlab® Functions for UAV Solution 3 230

Table E.1: Matlab® Functions for UAV Solutions 4 and 5.................................246

Table F.1: Matlab® Functions for UAV Solutions 6-8 256

Table G.1: Matlab® Functions for UAV Solution 9 275

List of Figures

Figure 1.1: A Set-Based Approach to Group Decision Making 2

Figure 1.2: A Point-Based Approach to Group Decision Making..........................2

Figure 1.3: A Simple Set-Based Sequential Design Process 5

Figure 1.4: The Affect of Set-based Design on Lead Time Estimations 6

Figure 1.5: A Set-Based Sequential Design Process 8

Figure 1.6: A Set-Based Sequential Design Process with Intervals 9

Figure 1.7: A Set-Based Sequential Design Process with Histograms 10

Figure 1.8: A Set-Based Concurrent Design Process with Regions 11

Figure 1.9: Arbitrarily Shaped and Disconnected Design Regions 13

Figure 3.1: Fully Dependent (left) and Fully Independent (right) BN's...............35

Figure 3.2: An Example Kernel Density Estimate...37

Figure 3.3: Parzen Window (left) and Naïve Bayes Classifiers (right). 39

Figure 3.4: The Effect of Standard Deviation on the Kernel Density Estimate 43

Figure 4.1: A Collaborative Design Problem for a UAV Wing 51

Figure 4.2. The UAV Design Problem with Merged Subsystems Teams 54

Figure 4.3. The UAV Collaborative Design Process..54

Figure 4.4. The Systems' Classifier for 10, 20, and 100 Training Points.............57

Figure 4.5. The Classifier's Affect on the Systems' Decision Boundary.............58

Figure 4.6. KBN Classifier Error Rates for the Systems' Design Space...............59

Figure 4.7. Interval Classifier Error Rates for the Systems' Design Space...........59

Figure 4.8. The Subsystems' Classifier for 10, 20, and 100 Training Points 61

Figure 4.9. The Classifier's Affect on the Subsystems' Decision Boundary 62

Figure 4.10. KBN Classifier Error Rates for the Subsystems' Design Space 62

Figure 4.11. Interval Classifier Error Rates for the Subsystems' Design Space ...63
Figure 4.12. The Loss Factor's Effect on the Systems' Decision Boundary.........64
Figure 4.13. KBN Classifier Error Rates for the Systems' Design Space 65
Figure 5.1. The Effect of the Standard Deviation on Classification of the Training Points..70
Figure 5.2. The Posterior Probability Decision Surface 75
Figure 5.3. The Proposed Method and Rule-Based Standard Deviations............78
Figure 5.4. The Proposed Method and LOOCV Standard Deviations 80
Figure 5.5. Systems Level Total Classification Error Rates 82
Figure 5.6. The Proposed Method and Rule-Based Standard Deviations............83
Figure 5.7. The Proposed Method and LOOCV Standard Deviations 84
Figure 5.8. Subsystems Level Total Classification Error Rates. 84
Figure 5.9. New Rule Standard Deviations 86
Figure 5.10. New Rule Total Error Rates 87
Figure 5.11. Decision Boundaries for $c = 0, 0.5, 1.0$ 89
Figure 6.1. The Systems' Classifier for 0, 10, 20, and 100 Simulation Runs.....100
Figure 6.2. KBN Classifier Error Rates for the Systems' Design Space 101
Figure 6.3. The Systems' Classifier for 10, 20, and 100 Training Points..........103
Figure 6.4. KBN Classifier Error Rates for the Systems' Design Space 103
Figure 7.1 Exploratory Samples for $N = 1, 2, 5, 10, 25$, and 100 Points 112
Figure 7.2 Different Methods for Producing Space-Filling Samples 113
Figure 7.3 The Systems' Classifier for 10, 20, and 100 Training Points.............119
Figure 7.4 KBN Classifier Total Error Rates for the Systems' Design Space 120
Figure 7.5 The Subsystems' Classifier for 10, 20, and 100 Training Points 120
Figure 7.6 KBN Classifier Total Error Rates for the Subsystems 121

Figure 7.7 Subsystems' Design Points within the Systems' Search Domain 122
Figure 7.8 Subsystems' Search Domain and Acceptable Region Classifiers 123
Figure 7.9 Subsystems' Search Domain and Acceptable Region Classifiers 124
Figure 7.10 KBN Classifier Total Error Rates for the Subsystems 125
Figure 7.11 Subsystems' Design Points within the Systems' Search Domain (left) and within the Systems' Acceptable Region (right) 125
Figure 7.12 Subsystems' Design Points Mapped to the Systems' Design Space 127
Figure 7.13 Subsystems' Search Domain and Acceptable Region Classifiers 128
Figure 7.14 KBN Classifier Total Error Rates for the Subsystems 129
Figure 7.15 Subsystems' Design Points within the Systems' Search Domain (left) and within the Systems' Acceptable Region (right) 129
Figure 8.1 CCN for Two Asymmetrically Coupled Design Teams....................135
Figure 8.2 CCN for Two Symmetrically Coupled Design Teams......................136
Figure 8.3 Merging CCN Nodes by Coordinating the Training Points 137
Figure 8.4 CCN with Bayesian Network Classifiers 138
Figure 8.5 Asymmetric Propagation of Class 140
Figure 8.6 Symmetric Propagation of Class 140
Figure 8.7 Parental Relationships without (left) and with (right) Common Design Parameters at the Subsystem Level...141
Figure 8.8 CCN without Common Design Inputs at the Subsystems Level........142
Figure 8.9 CCN with Common Design Inputs for the Co-parents 144
Figure 8.10 The UAV Collaborative Design Process...147
Figure 8.11 Subsystems' Search Domain and Acceptable Region Classifiers 148
Figure 8.12 KBN Classifier Total Error Rates for the Subsystems 149
Figure 8.13 KBN Classifier Error Rates for the Subsystems..............................149

Figure 8.14 Demonstrated Admissible CCN's 151

Figure 8.15 Admissible CCN Chains and Trees 152

Figure 8.16 CCN Polytrees 153

Figure 8.17 Propagation of Dependencies in a Hierarchical CCN 154

Figure B.1 Fuselage Parameters 181

Figure B.2 Wing Parameters 182

Chapter 1: Introduction

For many products, the design process is a complex system involving the interaction of many distributed design activities that need to be carefully coordinated. The extent to which a firm excels at this coordination has increasingly been recognized as a very important competitive advantage. Coordinating distributed design activities involves not only decomposing complex design problems and sequencing design activities, but also searching for satisfactory system-wide solutions. This research develops a new tool to improve one specific aspect of this challenge: quantitatively capturing a collaborative consensus of which designs are feasible options for meeting system-wide engineering requirements.

1.1 The Set-Based Collaborative Design Challenge

The design process is one of discovery, and resources must be allocated to balance consideration of a variety of design options with the development of at least one design in sufficient depth such that it can be produced and sold. The set-based philosophy is that by taking the time to understand the options and delaying the commitment to one design for as long as possible, there is less risk of redesign due to failing to meet internal requirements or customer expectations. The cost of redesign at later stages in the development process can be significant if many resources have been spent on work that will have to be redone. The evidence in support of the set-based philosophy began primarily with studies of Toyota's product development process, which is discussed in this opening chapter, along with other supporting research.

Set-based design principles are in practice informally every day because of their robustness with respect to group decision making. The underlying ideas are simple. If two or more people need to agree on something but are not likely to have the same constraints, then the set-based approach is to gather their feasible solutions and to find the intersection. This approach is illustrated in Fig. 1.1 where the horizontal axis represents the value to be agreed upon and the vertical axis represents a measure of preference, with zero representing infeasibility. In contrast, a point-based method might begin with the first person expressing his most preferred point, followed by the second person expressing her most preferred point, followed by iteration until a mutually feasible point is agreed upon. This approach is illustrated in Fig. 1.2.

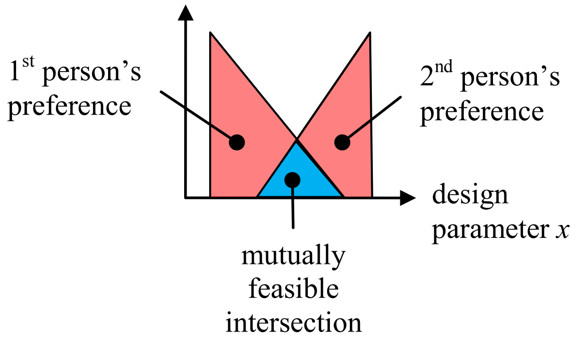


Figure 1.1: A Set-Based Approach to Group Decision Making

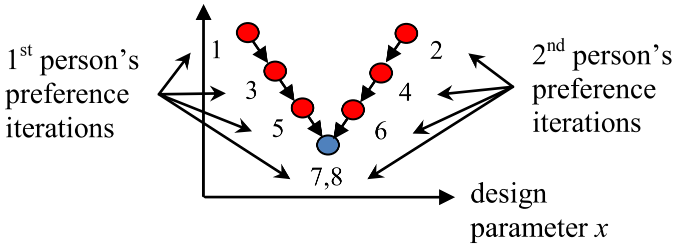

1st person's preference iterations
1
2
3
4
5
6
7,8
2nd person's preference iterations
design parameter x

Figure 1.2: A Point-Based Approach to Group Decision Making

Already some process trade-offs are identifiable. Eliciting someone's preferences over a design parameter's domain is more costly than generating single preference points only when needed. Hence, one can expect a set-based process to require more time up-front relative to a point-based process. So where is the time savings? In the set-based process, at the end of the negotiation, a set of options exists. When other people weigh in on the decision, they are likely to find a new mutually feasible region that is a subset of these options. However, when a new party enters the point-based negotiation, all of the previous stakeholders will probably need to iterate again in order to find a new mutually feasible design point. So, while the point-based approach appears to be resource frugal, it is subject to subsequent iteration and the more delayed the iteration, the more costly it becomes. A more complete picture of set-based design within a real product development process is discussed next.

In the 1990's, in-depth studies of Toyota's product design and manufacturing processes were conducted to understand how they achieved their remarkably low product development lead times (Ward et al., 1995; Sobek et al., 1999). Sobek, Ward, and Liker summarize what they call the 2nd Toyota paradox as: "Toyota considers a broader range of possible designs and delays certain decisions longer than other automotive companies do, yet has what may be the fastest and most efficient vehicle development cycle in the industry," (Sobek et al., 1999). They attribute this acgievement to what they call set-based concurrent engineering and identify its three underlying principles as:

1. Map the Design Space.
    - Define feasible regions.
    - Explore trade-offs by designing multiple alternatives.
    - Communicate sets of possibilities.
2. Integrate by Intersection.
    - Look for intersections of feasible sets.
    - Impose minimum constraint.
    - Seek conceptual robustness.

3. Establish feasibility before commitment.
    - Narrow sets gradually while increasing detail.
    - Stay within sets once committed.
    - Control by managing uncertainty at process gates. (Sobek et al., 1999)

The first two principles are the focus of this research, and, specifically, within these principles the problem of mapping the feasible regions and finding their intersections are addressed directly. At Toyota, it was noticed that each engineering department maintains a checklist that defines their guidelines to ensure feasibility. These checklists are shared among the collaborating teams at the beginning of a new development program. Because this knowledge is accumulated over many development programs, the initial cost of mapping the feasible regions is partially mitigated. The knowledge captured within the checklists can lead to very rapid and thorough decisions for many tasks. What the present research provides is a new checklist of sorts that can help teams more thoroughly define and communicate their feasible sets of designs. This research also provides two procedures for finding the intersection of the feasible regions.

To further fortify the claims that set-based design can reduce design process lead time despite the higher initial effort, Shahan and Seepersad (2010) conducted research to isolate how a simple set-based versus point-based design process can influence iteration. The reduction of iteration has long been a focus in the project management literature. Steward developed the design structure matrix as a tool to find a better sequencing of design tasks such that iteration loops have as few other tasks between them as possible (Steward, 1981). Researchers supplemented this early effort with more refined models that include a probability of iteration (Browning and Eppinger, 2002). Shahan and

Seepersad extended the model a step further by considering how the design process might affect the probability to iterate.

There are several potential causes for iteration between dependent design teams including errors, infeasibility, and convergence of a coupled analysis. Focusing on just iteration due to infeasibility, two designers conducting design tasks that are dependent such that one designer's results become the input to the other designer's simulation will iterate if the upstream designer chooses a design point that is infeasible to the downstream designer. In this relationship, the upstream designer has sole control over choosing the next design point. In a naïve throw-it-over-the-wall set-based strategy, the upstream designer will internally iterate until *n* design points are found that meet her local constraints and send these designs on to the second designer as shown in Fig. 1.3. With regard to the second designer who receives a set of *n* random design points, the probability that she will find that *m* of these designs meet her constraints, too, follows a negative binomial distribution. Thus the upstream designer can locally iterate longer to

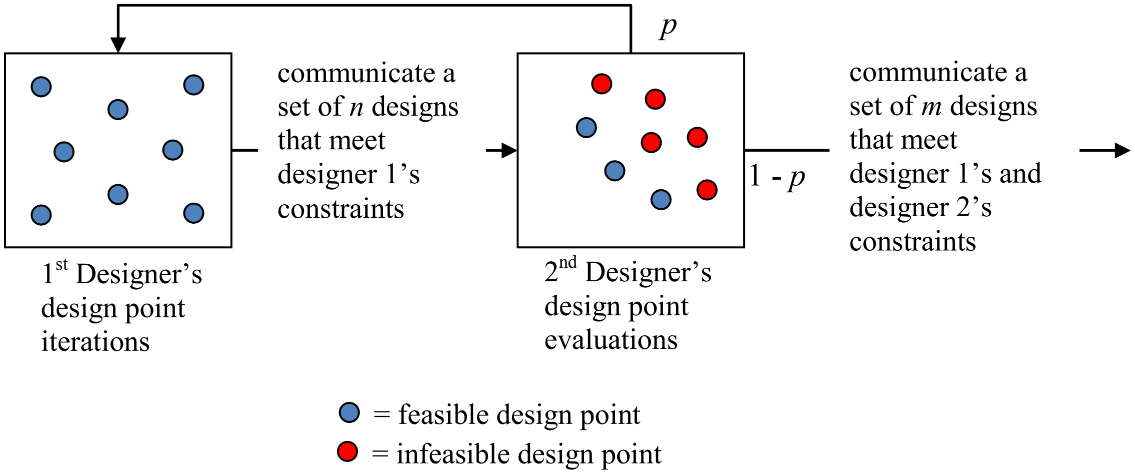

p
communicate a set of n designs that meet designer 1's constraints
1 - p
communicate a set of m designs that meet designer 1's and designer 2's constraints
1st Designer's design point iterations
2nd Designer's design point evaluations
● = feasible design point
● = infeasible design point

Figure 1.3: A Simple Set-Based Sequential Design Process

generate a larger set of designs (a larger *n*) and thereby reduce the downstream designer's probability of iteration, *p*.

Set-based design therefore provides designers with a mechanism to trade-off local iteration with iteration between design teams. However, this study also captured the potential inefficiency of set-based design that is the paradox described by Ward, et al. If the end of the design task is defined as discovering the first design that is feasible for all stakeholders and if there is no additional cost to iterating between teams, then a process can on average be shorter if the downstream check on feasibility of a design is made as soon as possible instead of waiting until a batch of *n* feasible designs is generated.

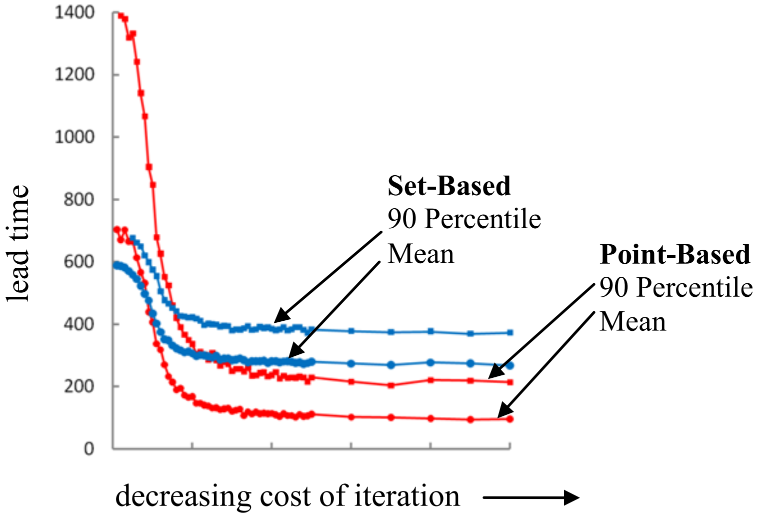


Figure 1.4: The Affect of Set-based Design on Lead Time Estimations

This effect was observed in the results of the study shown in Fig. 1.4 where the cost of iterating between teams is decreased from left to right on the horizontal axis and the strategy with the shortest lead time shifts from set-based to point-based accordingly (Shahan and Seepersad, 2010). The significance of the set-based concurrent engineering strategy rests upon the broader design context including the design environment. If factors exist that increase the cost of iteration between teams then set-based design becomes an important means to reducing project lead time. This cost could come from inefficiencies in communication, competition with other design tasks because of design resource constraints or, perhaps most importantly, from extensive rework due to cascading dependencies that were based upon using what turned out to be an infeasible design. However, the research in this dissertation provides a way to have the benefits of both a low probability of iteration due to set-based design as well as early feasibility feedback by using an efficient means for generalizing feasibility knowledge from a set of designs to regions of the design space.

The process just described is the classic throw-it-over-the-wall strategy that concurrent engineering seeks to avoid. The solution is to acquire guidance from downstream designers earlier in the design process. Sequential design processes could iterate faster with local gradient and constraint feedback such as typically occurs in distributed optimization frameworks. The design process could also be changed to include an initial study by the second designer to propose a target for the first design team to try to achieve. The first designer then iterates locally until she achieves the second designer's target or time runs out. However, these are point-based solutions that are prone to repeating because they do not seek knowledge in broader regions of the design space.

Extensions of point-based processes to reason about sets of discrete targets and design points have been proposed by Madhavan et al. (2008) in order to obtain a more complete understanding of the trade-offs within the collaborative design space as illustrated in Fig. 1.5. However, this approach only partially mitigates the knowledge generalization problem because it is not immediately clear how to extend the feasibility information from each point to the neighboring design space. If a future interaction requested the feasibility of a new design that did not exactly coincide with a prior point of the set, there is no clear mechanism for determining its feasibility. Extending the knowledge from a collection of points to regions of the design space allows designers to use their own simulation runs to map their feasible regions and query collaborators' maps to gain consensus on the system-wide feasibility of their choices. As long as a mutually feasible design space exists and the error of representing this region is acceptably low, the probability of iteration between design teams can be significantly reduced if not eliminated.

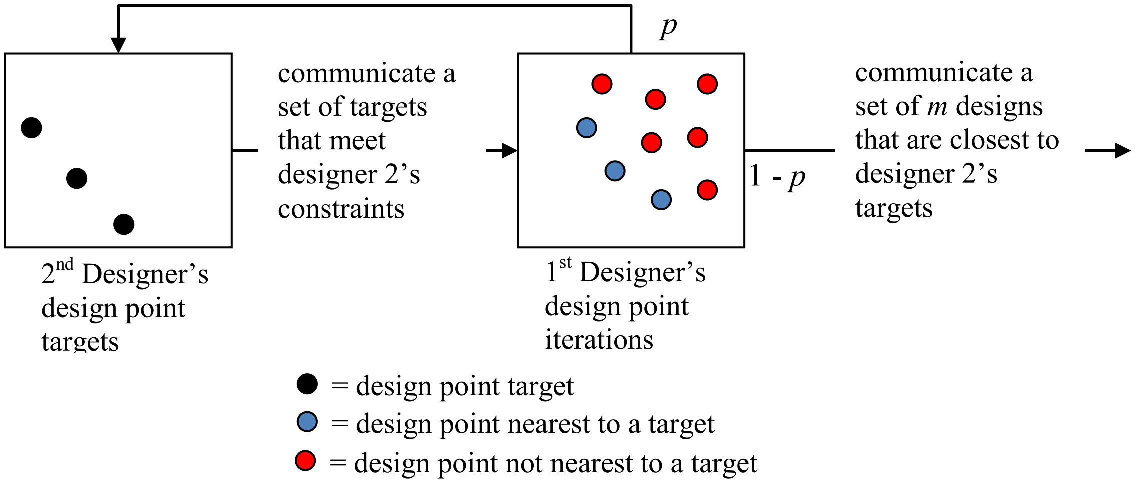

p
communicate a set of targets that meet designer 2's constraints
communicate a set of m designs that are closest to designer 2's targets
1 - p
2nd Designer's design point targets
1st Designer's design point iterations
● = design point target
● = design point nearest to a target
● = design point not nearest to a target

Figure 1.5: A Set-Based Sequential Design Process

Given a collection of feasible and infeasible points, how can this knowledge be generalized to judge if a new design is likely to be feasible or not? The standard answer from prior research is to use intervals to capture the knowledge of feasibility over a design region as illustrated in Fig. 1.6. However, the correct feasible region is also shown in Fig. 1.6 as having a diagonal boundary such as what might occur if decreasing both independent variables always improves the dependent performance parameter that must exceed a certain value or else be infeasible. The single interval will not accurately represent the diagonal boundary between feasible and infeasible designs. The first set-based design challenge is to provide a representation of regions of the design space that can accurately represent feasible regions better than intervals can. A quick fix would be to use many smaller intervals, dividing the space up into many smaller bins and to label each bin as being feasible or not as shown in Fig. 1.7. But a discrete design space imposes severe constraints on how the knowledge of a point's feasibility is extended to the design space. A more flexible and efficient representation will be introduced in Chapter 3.

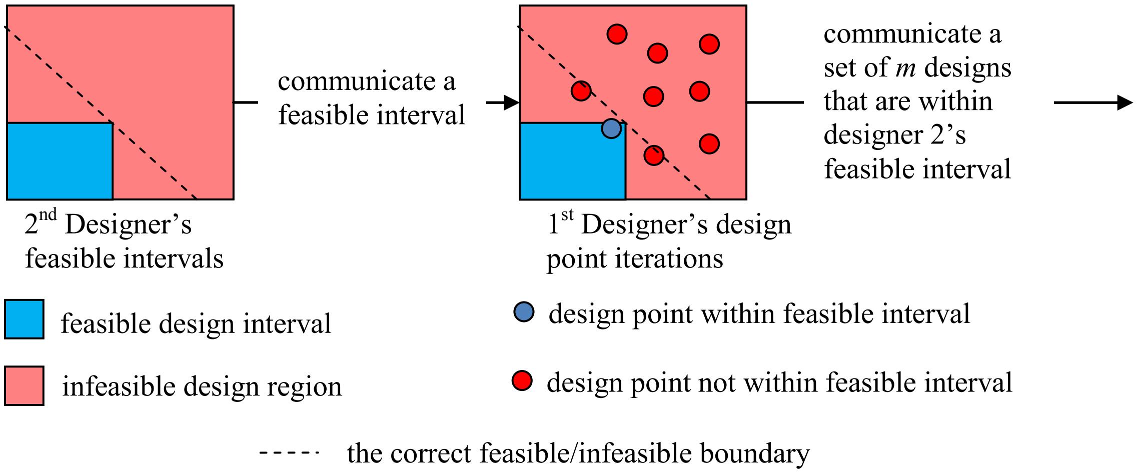

communicate a feasible interval
communicate a set of m designs that are within designer 2's feasible interval
2nd Designer's feasible intervals
1st Designer's design point iterations
feasible design interval
infeasible design region
design point within feasible interval
design point not within feasible interval
the correct feasible/infeasible boundary

Figure 1.6: A Set-Based Sequential Design Process with Intervals

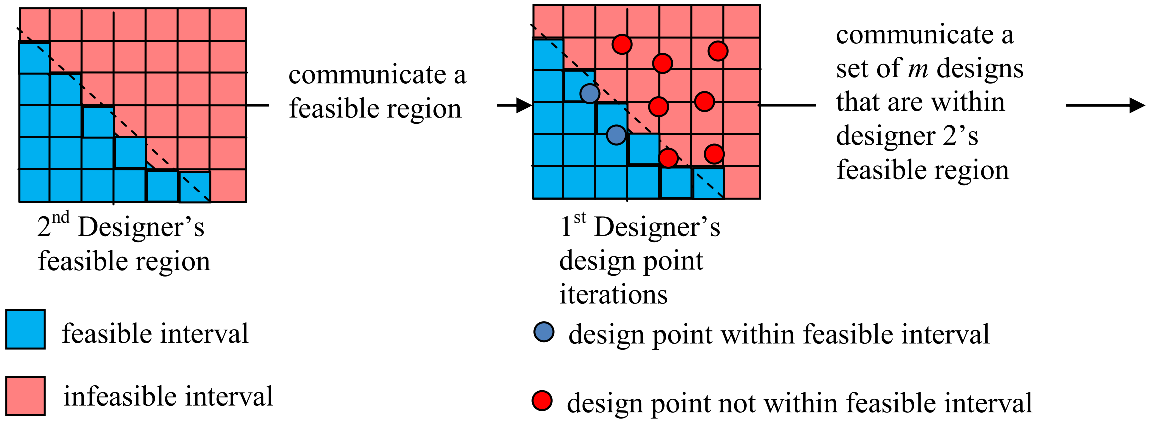

communicate a feasible region
communicate a set of m designs that are within designer 2's feasible region
2nd Designer's feasible region
1st Designer's design point iterations
feasible interval
infeasible interval
design point within feasible interval
design point not within feasible interval

Figure 1.7: A Set-Based Sequential Design Process with Histograms

The scenarios so far described have been primarily sequential in that the knowledge concerning feasibility is built up first and then used throughout the design process to ensure feasibility. If the generalization of knowledge from design points to design regions can be efficiently accomplished, more immediate feasibility feedback between concurrently collaborating designers can be provided as depicted in Fig. 1.8, making it possible to have both the set-based benefits of low iteration probabilities as well as the efficiency of resource allocation due to sampling only in the most mutually promising regions. For example, a designer will not need to sample their entire design space if the majority of it is clearly infeasible for their collaborator. The sooner each team has knowledge of their mutually feasible region, the sooner they can focus their resources on exploring that smaller region. The success of this ideal situation depends upon how fast the feasible design region can be accurately captured so that it does not significantly mislead collaborating designers. The second challenge of set-based collaborative design is to provide accurate feasibility feedback between collaborating designers as early as possible.

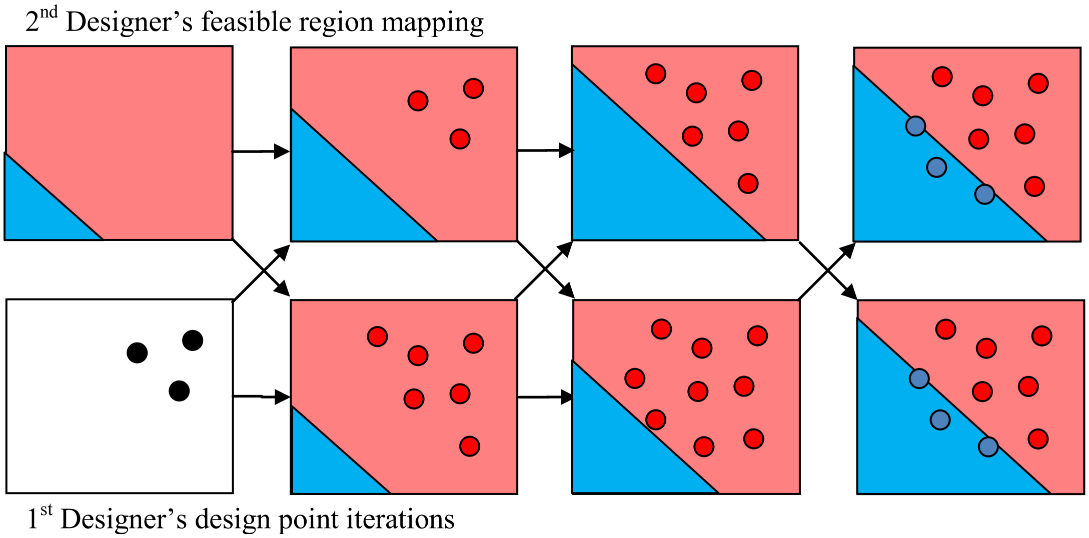

2nd Designer's feasible region mapping
1st Designer's design point iterations

Figure 1.8: A Set-Based Concurrent Design Process with Regions

In summary, a design tool is proposed that addresses two principles of set-based design: map the feasible regions of the design space and integrate collaborating teams of engineers by finding the intersection of their feasible regions. The perceived benefit is that collaborating teams of engineers will be less likely to iterate due to downstream discovery of a single design being infeasible, a situation project management research has sought to avoid. The potential reduction of late-stage iteration as well as the knowledge capture for facilitating future decision making will improve product development efficiency, decreasing lead times and potentially improving product quality. Furthermore, the efficient generalization of design feasibility knowledge from design points to regions of the design space might allow for concurrently developing and sharing maps of feasible regions such that resources can be quickly focused on the mutually feasible design region. The next section characterizes the proposed research in more depth.

1.2 The Research Scope and Requirements

The question remains: to what extent should one develop more alternative designs versus developing a more detailed understanding of one or more designs? Throughout a design process this question must be asked repeatedly and used to commit resources to further evaluation of a concept to determine its feasibility. The sooner a reliable judgment of feasibility can be made, the more efficiently resources can be allocated. Automation and simulation relieve some of the pressure to get high fidelity information earlier in the design process, and lower fidelity simulations are useful for rapidly eliminating concepts. The research in this dissertation does not address these broader aspects of the product development process. Specifically, it is assumed that simulations exist that can provide a uniformly accurate judgment of feasibility of a design alternative. It is also assumed that sufficient resources are available to search for feasible solutions and that the most influential independent and dependent design parameters have already been identified. These self-imposed constraints provide a reasonable scope for this research to present and advance the novel aspects of the proposed approach to set-based collaborative design. Within this scope, requirements for the proposed tool can be better defined.

The first requirement for this research, as motivated in the previous section, is to provide a method that is flexible enough to accurately describe the different shapes of feasible regions that might arise in collaborative design. Simple and common monotonic conditions will create a diagonal design space, as depicted in the earlier figures, which already challenges interval-based representations. How much more flexible then intervals should the proposed representation be? Simple contours can become quite complicated once they are mapped through nonlinear simulations. Inverse kinematic problems are known to lead to disconnected regions of the design space (Moore et al., 2009). The distributed design research has also recognized the need for representing disconnected regions of the design space (Liu et al., 2009). Without the benefit of an extensive empirical study of many design problems, one can set the goal high and then discuss any trade-offs that occur between the computational costs of more flexible representations versus their accuracy. Taking this latter approach, our objective will be to use a representation that can capture arbitrarily shaped and potentially disconnected feasible regions of the design space that do not lend themselves to being captured easily through simpler means such as intervals. Figure 1.9 illustrates some hypothetical examples.

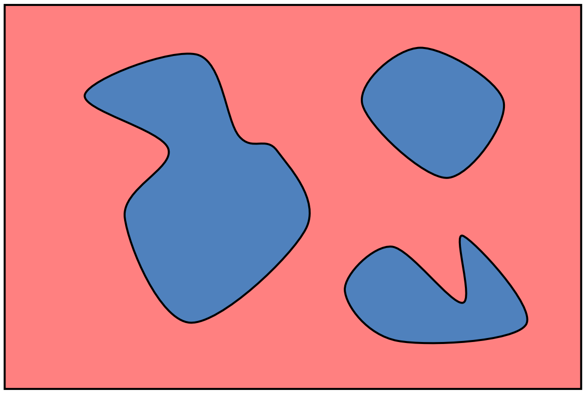


Figure 1.9: Arbitrarily Shaped and Disconnected Design Regions

Not all design problems and processes are complex enough to warrant the approach proposed in this dissertation. Specifically, there should be a significant gain to being able to provide another team with information such as: “If you decide to set this parameter to x_1, then we can achieve a feasible design over this range of values for y, but if you set the same parameter to x_2, then our feasible region changes to this, and if you ...” This information should not be trivially attained by evaluating a simple equation or following a few simple rules. Meeting this requirement will come with some overhead that will naturally need to be weighed against the potential gain, although every effort will be made to minimize any additional burden on resources. Furthermore, whenever the burden can be shifted from the designer to the computer without loss of effectiveness, it will be. The second requirement for this research is to automate the mapping process in order to minimize the burden on the designer as long as its effectiveness is not compromised. This requirement is in large part based upon the author's own experience in seeing design tools fail to be accepted by designers because they simply shift the focus of effort from the original problem to a new problem of using and maintaining the new tool without a clear gain. But this is not an argument in favor of just full automation.

Companies and the experts they employ have a specialized knowledge of their fields. This knowledge has hopefully helped to define some of their existing tools and processes to exploit the structure of the problem the company has decided to solve. If a design tool can leverage this existing knowledge to provide its service more effectively, it should. Evidence in favor of keeping designers in the loop during the search of the design space has been found in design steering studies (Carlson et al, 2008). In terms of exploiting knowledge of where and how to search the design space, the design tool should, if possible, accommodate knowledge from any source. This capability can be stated as an extension of the second requirement in terms of also being not automated to the extent that this will allow it to benefit from external information, such as experience-based knowledge of the designer. In this mode of operation, the tool will receive design points and knowledge of their feasibility from some external source and use it to produce its map of the feasible region. This is a minimal implementation. In order to still derive some benefit from the maps being developed, the tool would, at the request of its user, make sure its knowledge of design point feasibility is consistent with all collaborators' maps. The role of collaboration informs the final requirement discussed next.

In preliminary research, the author sought to construct a design environment for the express purpose of defining how set-based design support might further facilitate the collaborative design process. Student designers were required to design the wing of an unmanned aerial vehicle, a similar problem to the one used throughout this research and presented in depth in Chapter 4. The team was given the collective goal of finding a design that achieved a range over a certain limit as determined by a systems designer while also meeting the constraints of both the aerodynamics and structures designers. The designers sat around the same table, each with their own laptop, and they were given instructions not to share their simulations with other designers. A successful team provided an email trail of simulation input and output results such that all shared parameters between the designers agreed in value and the range met or exceeded the requirement.

The design processes that arose within the given experimental context all exhibited a lot of verbal collective reasoning about what would make a good design. Each designer's activity proceeded concurrently, balancing their attention between the competing threads of learning more about their problem, communicating with the other designers about how to best proceed, and testing the performance of each other's designs. The design tool should support such a concurrent and interactive environment. The goal should be to provide the collaborating designers expedient and concurrent feedback on the feasibility of their design choices with respect to their collaborator's constraints. Achieving this goal removes the cost of communication from the designer's attention and provides the designer collaborative feasibility information earlier in the design process.

The third requirement for the proposed design tool is to support designers working in parallel who will develop and share their feasibility knowledge as they desire.

Table 1.1: Research Requirements

1	Communicate arbitrarily shaped and potentially disconnected regions of the design space.
2	Provide a range of automation options from full to no automation of the mapping process.
3	Support designers working in parallel who can develop and share their maps as desired.

The proposed requirements for this research are summarized in Table 1.1. These requirements should be understood as goals to be striven for and that hence have informed the direction this research took in development of the proposed tool. The extent to which the proposed method meets these requirements is discussed in depth throughout the dissertation. Coming closer to achieving these goals sets the proposed method apart from existing methods as discussed further in the literature review of the next chapter.

1.3 The Research Hypothesis and Overview

In light of the requirements set forth in the preceding section, the research hypothesis can be stated:

Bayesian networks can be used to facilitate distributed design by providing for a set-based and parallel process. The resulting methodology will be set-based in that arbitrarily shaped and potentially disconnected regions of the design space are communicated between collaborating designers. The resulting methodology will also be *parallel in that Bayesian networks can be independently developed and shared as desired during the design process. Furthermore, the resulting methodology has the potential for anything from full automation to no automation of the mapping process.*

This research began with a significant amount of work that led to the requirements and hypothesis as developed in this chapter. The extent to which these requirements present a novel advancement to the prior collaborative design research is discussed in depth in the second chapter. The third chapter presents the underlying technology that was selected, Bayesian network classifiers (BNC), to meet these requirements and discusses any fundamental limitations or additional challenges the technology presents in order to meet the stated requirements. The fourth chapter presents a collaborative design problem that is used throughout the research as a testbed to judge the extent to which the proposed method meets these requirements. A baseline solution to the example problem is also presented in Chapter 4 to provide a point of comparison for the methods developed in Chapters 5-7. Chapter 5 presents an important advancement to BNC's that eliminates some mapping errors. Chapter 6 presents and validates the exploitation of expert knowledge for significantly improving the efficiency of the BNC. Chapter 7 presents additional search capability that is facilitated by the choice in technology and enhances the automation of the process. Chapter 8 presents a new method for finding the intersection of three designers sharing their maps of their feasible regions. The dissertation closes with a discussion of the extent to which the requirements within the hypothesis were successfully met and important directions for future work.

Chapter 2: The Collaborative Design Research Context

There are several threads of research addressing the needs of set-based collaborative design and an attempt is made in this chapter to trace them and to understand their relationship to the work presented in this dissertation. By doing so, the contributions of the proposed research will be better understood. Not all of the differences between the proposed approach and the prior research are described here, only the most fundamental differences with respect to the broader goals of the collaborative design problem as presented in the opening chapter. Throughout this document, other differences are mentioned within the specific context of each chapter.

The review of existing methods will be informed by the principles developed in the opening chapter. First, the method is to be set-based in terms of generalizing design knowledge to promising regions of the design space that can be arbitrarily shaped and potentially disconnected. Second, the method used to capture the knowledge should support local and concurrent development by designers who will share their knowledge with their collaborators in order to find a mutually feasible design space. Third, the generalization process should be automatable, given a means of determining the correct classification of a design point. The method should also support interactive use by which the designer can influence the mapping process in order to exploit expert knowledge. This chapter is organized around the first principle with the prior work categorized by increasing expressiveness. Methods that have addressed one or more of the other principles in a significant way will also be identified.

2.1 Points and Point Sets

The line of research generally called multidisciplinary design optimization (MDO) is characterized by the adaptation of centralized optimization techniques to a decentralized framework. The motivation for the approaches comes from the desire to apply optimization techniques to large, system-wide design problems without integrating all of the design tasks into a single centralized design problem and accompanying analysis, which would be highly computationally expensive. For many MDO methods, process focus typically alternates between a coordinating group and several disciplinary subgroups which are given varying degrees of control. This framework allows for parallel processing among the disciplines in between iterations with the coordinating group. Many instances of MDO are point-based in that the hand-off of process focus occurs with the communication of a single value for each coupled design parameter and its derivatives—if required—between the disciplines and the coordinating group. Concurrent subspace optimization (CSSO) (Sobieski, 1988), bi-level integrated system synthesis (BLISS) (Sobieski et al., 2000), collaborative optimization (CO) (Kroo et al., 1994) and enhanced collaborative optimization (ECO) (Roth and Kroo, 2008) are primary examples of point-based MDO. Analytical Target Cascading (ATC) is a noteworthy point-based distributed product development method that coordinates hierarchical dependencies between design teams using system level targets and weighted sum objective functions to encourage teams to converge upon the same design (Kim et al., 2003). In contrast to many MDO methods, ATC includes a proof of convergence for a nested procedure that recursively coordinates subsystem activity at a system level using targets in a hierarchical dependency structure, subject to the problem space being convex and continuous (Michelena et al., 2003).

MDO as a discipline has evolved to adapt more global and set-based optimization methods to the coordination of distributed design activity. Genetic algorithms (GA) are set-based in that a population of design options is used to evolve increasingly optimal solutions (Holland, 1975; Goldberg, 1989). In coevolutionary genetic algorithms for

MDO (CMDO), disciplines are represented by species and competition within a species proceeds according to standard genetic algorithms (Nair and Keane, 2002). The species interact through their coupled variables by posting their top performing values and using their collaborator's posted top performing values in a combinatorially exhaustive evaluation of their current generation. This method is notable for achieving an asynchronous, parallel process in the same spirit as the third principle of this research. Particle swarm optimization (PSO) works with a set of designs that are each incrementally improved based upon local information as well as feedback from the collective swarm (Kennedy and Eberhardt, 1995). Particle swarm has been successfully applied to the MDO of an aircraft wing where one discipline handles the aerodynamics and major geometric parameters while the sub-discipline optimizes the smaller scale structural parameters to minimize weight (Ventner and Sobieski, 2004).

In another variant of applying genetic algorithms to MDO, Gunawan et al. propose an entropy-based multiobjective multidisciplinary genetic algorithm (E-MMGA) for encouraging solution diversity at the coordinating level and then using the results to seed the parallel disciplinary multiobjective genetic algorithm analyses (Gunawan et al., 2009). The results of the design groups are synthesized into a large set of solutions from which the next seeds for the design groups are chosen by the coordinating group to maximize the population diversity. The iterations between the coordination group and the design groups repeat until there is no improvement in entropy. The entropy is calculated using a density function that is the aggregate of influence functions centered on the design points. The density function is similar to the kernel density estimate used in this research although the emphasis in this research is primarily on classification and not just indentifying designs on the Pareto frontier. Future work could investigate using an entropy estimate to supplement the search methods developed in Chapter 7.

Metamodels, also known as surrogate models, have also been applied to multidisciplinary design optimization with many references provided in the summary by Sobieski and Haftka (1995). Because metamodels interpolate the output of simulations based upon a discrete set of design points, they can be independently developed and shared for the purposes of concurrent distributed design. In this sense, metamodels generalize design results from sets of design points to design regions and then communicate results over the entire design domain. Metamodels however are an approximate stand-in for the analyses and they still need to be explored, and this exploration still needs to be coordinated by some method. For example, Batill et al (1999) use metamodels to facilitate CSSO, one of the founding MDO methods that was first developed without metamodels. More discerning uses of metamodels will be discussed later in the context of applications of game theory to collaborative design.

This concludes the review of MDO for the purposes of set-based collaborative design. These methods are predominantly optimization-oriented and automated with additional overhead associated with coordinating the convergence of the distributed design activity. The set-based characteristics are hence a result of optimization methods that develop a population of design possibilities such as GA's and PSO. These population-based optimization methods only support the goals of set-based design to the extent that they are multi-objective and attempt to describe trade-offs within the design space as opposed to converging upon a single optimum. There is minimal knowledge capture for future use if the only result of the process is a single design point.

A primary example of a method designed specifically to uncover the tradeoffs in a design problem with competing goals is the compromise decision support problem (cDSP) (Mistree et al., 1993). Within this framework, collaborators exchange goals for shared parameters that are traded off against local goals and that are subject to meeting local constraints. With a judicious choice of goals, a Pareto optimal set of designs can be found such that improving achievement of any one goal comes at the expense of less achievement of any other goal. One of the strengths of the cDSP is the wide applicability of the framework to several different approaches. Two such approaches that are notable for their contribution to collaborative design are reviewed next.

Lewis and Mistree (1998) present a game theoretic model of collaborative design that hinges upon constructing and sharing rational reaction sets (RRS). A designer's RRS is a metamodel that represents his best choice for the parameters that he is responsible for as a function of design parameters that are shared with other designers and effect his results. When designers share RRS's they are in effect sharing a region of the design space defined over shared parameters. An RRS represents more processed information than typical metamodels or simulations; it represents a designer's preferred objectives mapped back onto a design domain. An RRS summarizes design knowledge that would be prohibitively expensive to represent if it were not approximated using metamodels. The result of this approach however is by itself not particularly set-based because a designer uses another designer's RRS by submitting a choice for his/her design parameters and getting back their collaborator's best option, in the form of a single point. However, by iterating over one's own design options and obtaining one's collaborators' choices, tradeoffs can be mapped out with a set of points that otherwise might require extensive iteration between teams. A notable variation of the same principle includes concept selection from a system's perspective (Malak and Paredis, 2007)

The cDSP has also been used for the purposes of set-based collaborative design in a framework called the set-based method (SBM) wherein a set of targets are generated at the system level that become subsystem goals (Carlos et al., 2006; Madhavan, 2007; Seepersad et al., 2007; Madhavan et al., 2008). The subsystem teams tradeoff local goals and goals on shared parameters in order to communicate a Pareto set of design options to the system level design team. However, as discussed in the first chapter, because this approach does not generalize the knowledge from the set of points to regions of the design space it may not be clear if a new point will be acceptable or not. With the exception of metamodels, the same limitation applies to all of the other methods reviewed in this section. Even though metamodels interpolate the design space they still require other means for defining regions of the design space. The most common way of moving beyond this limitation is to use intervals, the subject of the next section, to reason about regions of the design space.

2.2 Intervals

Interval analysis gained recognition in the United States through the work of Moore (1966). The original goal of interval analysis was to trace the propagation of error or variation through calculations from intervals over the independent variables to intervals over the dependent variables within which the correct answer could be guaranteed to lie. Because these errors are assumed to be independent variations on each input, multiple occurrences of a variable within a calculation each contribute to the resulting breadth of the output's interval which can hence be a function of the form of the calculation. Example applications include numerical rounding errors and worst-case tolerance stack-ups. The power of interval analysis methods is their guaranteed enclosure of the desired result and their expedience that comes from determining interval propagation rules through commonly used functions.

A notable extension of interval methods to mechanical engineering design is the Method of Imprecision (MoI), based upon fuzzy set principles (Wood and Antonsson, 1988). The MoI extends interval methods by including a fuzzy measure of preference between zero and one defined over each design parameter. Fuzzy preferences are propagated through the design calculations into the performance space by using interval analysis methods for every interval defined by a different level of preference. A couple of the strengths of the MoI is the inclusion of preference and the careful consideration of methods of propagating and combining preferences that adhere to well defined properties (Wood et al., 1989; Otto and Antonsson, 1991). The applicability of the MoI to set-based collaborative design was recognized and researched primarily in terms of how different preferences between teams can be negotiated and trade-offs explored (Antonsson and Otto, 1995; Scott and Antonsson, 1996). For example, the property of annihilation is important because each team must have the ability to veto a design that does not meet their constraints. However, the MoI did not consider the dependencies between design parameters, resulting in only hypperectangular representations of the feasible regions. Although it should be recognized that fuzzy methods could be employed for the purposes of representing more complicated regions of the design space.

Ward's mechanical design compiler (MDC) research began prior to his involvement with the studies of Toyota's set-based product design process although it already showed signs of his preference for reasoning about regions of the design space in its extensive use of intervals (Ward, 1989). The two main strengths of the MDC are the extension of interval methods to different mechanical design modes of reasoning, called the labeled interval calculus, and applying the calculus in the form of rules that reduce the set of possible choices (Ward et al., 1995; Finch and Ward, 1997). In the context of collaborative set-based design, Chang and Ward (1995) propose a decentralized and concurrent approach where designers communicate the marginal cost of the shared parameters that they control and seek robust solutions that are insensitive to variations in the shared parameters that they do not control. This is called conceptual robustness.

Conceptual robustness has also been applied in the game theoretic approach using robust design methods to make the leader's decisions insensitive to the variation of the followers design variables, providing greater design freedom for the follower (Chen and Lewis, 1999). The interval-based constraint satisfaction (IBCS) method also extended the game theoretic approach for set-based collaborative design (Panchal et al., 2007). Each designer in the IBCS method uses arc and box consistency from interval analysis to sequentially reduce the intervals, gradually eliminating incompatible regions of the design space.

Set-based approaches that use intervals have also been proposed that take a systems perspective on the collaborative design problem. Recent work by Liu et al. (2007) uses the quantization algorithm to divide the system level design space into hyperrectangle domains called ranged sets. They propose the communication of the ranged set that is the most achievable by subsystems as determined by a flexibility metric. Interestingly, the flexibility metric is determined by integrating an aggregation of achievability functions centered on the sample points. As will be shown in more detail in Section 3, this is similar to the kernel probability distribution used in this research. However, Liu et al. do not propose communicating the aggregated achievability functions, and instead propose communicating the ranged set as the targets.

Other interval methods consider how to choose between concepts that have a range of performance due to imprecision in the early stages of design. Wood and Otto (1995) developed a method for propagating probabilistically characterized uncertainty and an associated confidence metric to help define the extent to which a concept outperforms another according to the resulting probability distributions over ranges of a performance parameter. Probabilistic methods are also used by Malak et al. (2009) to reason in a set-based manner about either eliminating or further exploring concepts based upon ranges of system level performance parameters. The proposed approach does not presently consider the systems perspective of needing to choose between concepts as in the previous two examples and takes instead a subsystems perspective of mapping system level requirements to the subsystem design space. The systems perspective will be left for future work.

Many of the interval-based methods reviewed in this section take the perspective of propagating ranges of values defined over input parameters through simulations in order to find the range of values a performance parameter can take on. This perspective will be called the forward propagation approach. In contrast, this research seeks to find the regions of the input domain that satisfy interval constraints on the outputs of simulations, called inverse propagation. The challenge in forward propagation is to find combinations of input parameters, which are typically allowed to independently take on different values over predefined intervals, that define the limits of achievable output values. As will be seen in Chapter 4, the hyperrectangular regions of independently applied intervals, also known as boxes, are not well suited to capturing the more complex regions that result from the inverse propagation of intervals through potentially nonlinear simulations. The labeled interval calculus and ICBS have taken a step toward considering how dependencies between parameters affect the mutually feasible region of the design space however they both still adhere to a box representation of feasibility. The next section presents methods that are able to represent nonrectangular regions of the design space, a necessary capability for more accurately reasoning about the inverse propagation of intervals. As an aside, the methods in this section do not directly address automation or concurrency of the collaborative design process with the exception of ICBS which is a sequential process.

2.3 Moving Beyond Parameter Independence

More sophisticated interval methods have been developed that can represent arbitrarily shaped and potentially disconnected regions of the design space. For example, collaborative design using solution spaces (CDSS) uses a 2^k-tree data structure to represent hyperrectangular subdivisions of the design domain where a division occurs when any of the 2^k corners of a subdivision straddles the boundary between feasible and infeasible designs (Lottaz et al., 2000). However the sampling sequence used to explore the design space in the proposed method is considerably more flexible than the 2^k-tree data structure used in CDSS that requires placement of the samples in the corners of each hyperrectangular subdivision. In the proposed method, designers that prefer to choose the next design point will be able to do so freely and still have the design point be usable by the classifier.

Joint probability distributions as encoded in a Bayesian network have also been used for simulation-based design to represent the feasible region of the design space subject to applied constraints (Ivezic and Garret, 1998). The joint probability distribution was visualized using histograms in an agent-based distributed software framework for the concurrent design of a frame structure. The software gives designers visual feedback on the feasibility of their design decisions, providing conflict identification and resolution capabilities. The approach is called a simulation-based decision support system (SB-DSS) and it shares many of the features of this research with some important exceptions. The SB-DSS approach uses a neural network to approximate the simulation results and then samples the network to generate the design points for the feasibility histogram in a fully automated initial step. The interaction with the designer comes after the feasible region has been mapped and a final design choice is identified by narrowing intervals of the parameter values. In contrast, this research addresses more directly which designs to evaluate, placing the designer interaction directly in the sampling loop as opposed to later in the final design selection process. By providing for designer interaction with the search process, expert knowledge can be incorporated to guide it.

In another application of Bayesian networks to engineering design, Xiang et al. (2004) propose using collaborative design networks (CDN). CDN's are actually influence diagrams that are an extension of Bayesian networks to include deterministic parameters and measures of utility. A portion of the nodes and their connecting edges represent the feasibility constraints on the design parameters and their dependencies that are necessary to represent more complicated regions of the design space than the hyperrectangular regions represented by intervals. CDN's are also multiply sectioned Bayesian networks (MSBN's) that divide the design space into regions of responsibility for distributed solving in an agent-based framework. The solution to a CDN is the design with the maximum expected utility. However, no procedure is provided to construct a CDN or its encoding of the feasible region. The proposed research provides this capability and could be seen as a precursor to using a CDN to reason about preferred regions of the design space and not just feasibility.

Design steering more closely achieves the goal of the proposed research for designer guided search. Design steering uses point sets and histograms among other tools to visualize regions of the design space (Carlson et al., 2008). However, the histograms are used only as a visualization tool for feedback to the designer and hence are limited to 2D and 1D marginal distributions. Furthermore, design steering is not presented as a collaborative design tool so there is no attempt to use the histograms to determine a new point's feasibility.

2.4 Discussion

The methods reviewed in this chapter all have unique perspectives and represent important contributions within their context. Within the context of this research, they all have limitations with respect to representing the feasible regions of the design space that result from the inverse propagation through simulations of interval requirements for performance parameters. For those methods that are based on a discrete set of design points, there is no ability to judge a new point's feasibility. Interval-based methods are limited to representing only hyperrectangular design regions. The remaining methods, with the exception of design steering, have the ability to represent arbitrarily shaped and potentially disconnected regions but are limited primarily in their lack of support for integrating the designer into the search process. One common aspect of these remaining methods is that they assume an initial seeding of the design space in order to represent the design region of interest. They focus on providing tools to explore the resulting design region as defined by this initial set. In contrast, the focus of the proposed method is to elicit guidance from designers during the creation of the set that is simultaneously used to more accurately define the regions of interest, combining the knowledge capture with the design space exploration and therefore providing a more expedient design process. Chapter 6 demonstrates the possibility of using designer knowledge to improve the efficiency of representing regions of the design space.

Furthermore, the technologies used by these remaining methods, including design steering, represent the feasible region of the design space with smaller and smaller interval divisions. In the limit of more subdivisions with ever smaller interval regions, these methods will be able to map arbitrarily shaped and potentially disconnected regions of the design space. This approach will not scale well if a grid is used to subdivide the design space because the number of bins grows exponentially with increasing dimension.

Bayesian networks and the 2^k-tree can both help mitigate the exponential growth. However, discretization of the design space enforces upon every design point an unnatural generalization to a rectangular region with a center that is not on the point. Much more flexible methods exist that more naturally generalize feasibility knowledge from single points to the design space. Furthermore, prior methods do not exploit knowledge from infeasible points. Methods called classifiers are more flexible and also use the infeasible points. One particularly flexible classifier that is used in this research is introduced in the next chapter.

Chapter 3. The Kernel-Based Bayesian Network Classifier

As motivated in the first chapter, the primary use of classifying a design space in terms of acceptability is to help a collaborator decide if a new design point will be acceptable or not without running the new design point through a time-consuming simulation. The classifier takes a design point as an input and returns the class label: feasible or infeasible. In this chapter, we present the details of a classifier that uses Bayesian networks (BN) (Pearl, 1988) and kernel density estimation (KDE) (Parzen, 1962; Silverman, 1986; Scott, 1992) in order to create probability distributions that interpolate the known acceptability of design points, called the training points, to new design points whose acceptability is uncertain. The use of probability distributions also achieves a secondary use of design space classification: to generate new candidate design points with a high probability of being acceptable to all collaborating parties.

Using probability distributions for classification has a theoretical foundation in Bayesian decision theory which is covered in Section 3.1. Section 3.2 formulates how Bayesian networks and kernel density estimation are combined to create a very flexible model of probability distributions for use in the proposed classifier, called the kernel-based Bayesian network (KBN) classifier (Perez, et al., 2009). Sections 3.3 and 3.4 discuss some practical details of using the KBN classifier. Section 3.5 discusses the classifier's representation capability. The final section discusses classification in light of the needs of this research, needs which are briefly touched upon next.

In choosing a classifier for this research, two primary classifier properties were sought. First, the classifier should have a richer representation than intervals. As will be demonstrated in Chapter 4, intervals have a very limited representation capability for the acceptable regions of design spaces. Ideally, the classifier for this research would be able to accurately represent any arbitrarily complex region of the design space asymptotically in the number of training points, and the modeling error would decrease rapidly as more training points are acquired.

The second necessary classifier property for this research involves the ability to generate new design points from the mutually acceptable design space of all collaborators. For this reason, probability distributions have been chosen to represent the acceptable regions because they can be sampled. With enough training points, the samples will have a high probability of also being acceptable. Being able to generate new and acceptable design points allows for a focusing of resources upon the important regions of the design space using adaptive sampling methods that are developed in Chapter 7.

3.1 BAYESIAN DECISION THEORY

Using probability distributions for classification has a theoretical foundation in Bayesian decision theory. This section presents these foundations based upon (Dudda and Hart, 2001). Consider a two category classification: c_1 to represent the satisfactory region of the design space and c_2 the unsatisfactory region. The classification is over a bounded D-dimensional design space for which a single design instance can be represented by a vector, $\boldsymbol{x} = [x_{i=1\ldots D}]^{\boldsymbol{T}}$. If we can express the class conditional probability of a design instance given a category, $p(\boldsymbol{x}|c)$, then Bayes formula can be used to find the probability of the class given design parameters, $p(c|\boldsymbol{x})$**,** according to Eq. 3.1.

$$P(c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|c)P(c)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|c)P(c)}{\sum_{k=1}^{2} p(\boldsymbol{x}|c_k)P(c_k)} \tag{3.1}$$

Design $\boldsymbol{x}$ is classified as a member of class c_1 and not class c_2 when $P(c_1|\boldsymbol{x}) > P(c_2|\boldsymbol{x})$. It follows that we can ignore $p(\boldsymbol{x})$ to get Eq. 3.2 as the rule for assigning class membership.

$$\text{Decide } c_1 \text{ if } p(\boldsymbol{x}|c_1)P(c_1) > p(\boldsymbol{x}|c_2)P(c_2) \tag{3.2}$$

It can be shown that the optimal decision boundary for minimizing classification error is when the two sides of Eq. 3.2 are equal. Risk in terms of loss factors, λ_{ij}, associated with deciding that the design point belongs to class c_i when the correct classification is class c_j can be incorporated as shown in Eq. 3.3 from which we see that losses from decisions that result in misclassification rescale the two sides of Eq. 3.2. The effect of this rescaling on the performance of the classifier will be demonstrated for an example design problem in Chapter 4.

$$\text{Decide } c_1 \text{ if } \lambda_{21}p(\boldsymbol{x}|c_1)P(c_1) > \lambda_{12}p(\boldsymbol{x}|c_2)P(c_2) \tag{3.3}$$

If the data exists to effectively approximate the class prior probabilities, $P(c)$, then the class conditional probabilities of $\boldsymbol{x}$ given c, $p(\boldsymbol{x}|c)$, can be approximated, otherwise the posterior probabilities of c given $\boldsymbol{x}$, $p(c|\boldsymbol{x})$, should be directly approximated or the effect of the priors should be removed by setting them equal. In this research, the prior probabilities are estimated using the frequencies of each class according to Eq. 3.4 where the padding of a single observation of each class helps to moderate the approximation at very low sample sizes. N is the total number of training points, and N_c is the total number of training points for class c.

$$P(c) \cong \frac{N_c + 1}{N + 2} \tag{3.4}$$

For the task of approximating the class conditional probability distribution, $p(\boldsymbol{x}|c)$**,** we next describe Bayesian networks and Gaussian kernel probability distribution estimation.

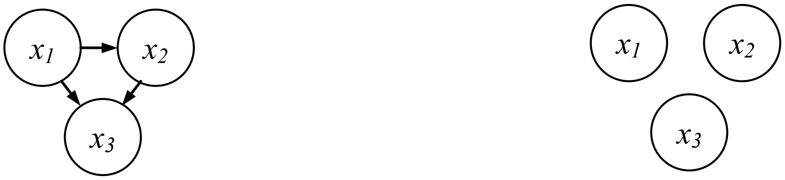
3.2 KERNEL DENSITY ESTIMATION AND BAYESIAN NETWORK CLASSIFIERS

The formulas that combine Bayesian networks and kernel density estimation are presented in this section, resulting in a very flexible model of joint probability distributions that can be used to approximate the class conditional probability used in classification. The development of the formulation follows the results of several researchers: (John and Langley, 1995; Hoffman and Tresp, 1996; Bosman and Thierens, 2000).

Bayesian networks (BN) encode a factored joint probability distribution (PD) as a directed acyclic graph (DAG) where the edges from the parent nodes to a child node mean that the child node's probability is conditionally independent of its non-descendants, given its parent nodes (Pearl, 1988). In other words, setting the values of a node's parents makes that node dependent only upon its descendent nodes, i.e. the nodes that are reachable following any chain of arcs from that node. Hence, using the definition of conditional independence, a BN represents a joint PD in the factored form shown in Eq. 3.5, as long as the variables are numbered in topological order, meaning that all ancestors of a variable have lower numbers and all descendents of a variable have higher numbers. The notation $\boldsymbol{pa}_i$ will be used to denote the *K* variables, x_k, that are the parents of variable x_i.

$$p(\boldsymbol{x}) = \prod_{i=1}^{D} p_i(x_i|\boldsymbol{pa_i}) \tag{3.5}$$

The root nodes of the graph, which have no parents, have probabilities that are simply $p_i(x_i)$. Figure 3.1 shows two extreme examples of BN's: the fully dependent joint PD represented by a fully connected DAG on the left, and the fully independent joint PD represented by the empty DAG on the right.



$p(\boldsymbol{x}) = p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$

$p(\boldsymbol{x}) = p(x_3)p(x_2)p(x_1)$

Figure 3.1: Fully Dependent (left) and Fully Independent (right) BN's

By having a formula that calculates the conditional probability of a variable that is dependent upon the value of its parent variables, the joint PD can be evaluated according to the topological order of the graph, beginning with the root nodes and proceeding to their children, and then evaluating their children's children until the leaf nodes are reached. A particular instance of a variable, x_i, will be denoted by $\hat{x}_i$. The j^{th} instance out of N total known design points will be expressed as $\hat{\boldsymbol{x}}^j$. Using this notation, Eq. 3.6 expresses the conditional probability that is calculated at each node of the graph in terms of the already evaluated K parents of variable x_i for the current j^{th} instance.

$$p_i^j\left(x_i|\boldsymbol{pa}(x_i) = \hat{x}_{k=1\ldots K}^j\right) = \frac{p\left(x_i, \boldsymbol{pa}(x_i) = \hat{x}_{k=1\ldots K}^j\right)}{p\left(\boldsymbol{pa}(x_i) = \hat{x}_{k=1\ldots K}^j\right)} \tag{3.6}$$

Kernel density estimates (KDE), also known as Parzen windows, are used in this research for calculating the conditional probabilities at each node of the BN (Parzen, 1962; Scott, 1992; Silverman, 1986). KDE's center a function, the kernel, at each of *N* design points. The magnitude of the kernel is a function, $\mathcal{K}$, of the distance from a point in the design space, $\boldsymbol{x}^j$, to the design point at its center, $\widehat{\boldsymbol{x}}^j$, as well as kernel parameters, $\boldsymbol{\theta}$. The probability density estimate for each point in the design space is the average of the influence of all *N* kernels, according to Eq. 3.7.

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{j=1}^{N}\mathcal{K}\left(\boldsymbol{x},\widehat{\boldsymbol{x}}^j,\boldsymbol{\theta}\right) \tag{3.7}$$

A weighted average can also be used according to Eq. 3.8.

$$p(\boldsymbol{x}) = \sum_{j=1}^{N} w^j\mathcal{K}\left(\boldsymbol{x},\widehat{\boldsymbol{x}}^j,\boldsymbol{\theta}\right), \quad \sum_{j=1}^{N} w^j = 1 \tag{3.8}$$

A common kernel function used in probability distribution estimation is the *D*-dimensional normal distribution of Eq. 3.9 with a diagonal covariance matrix. Gaussian kernels are used exclusively throughout this research.

$$\mathcal{N}(\boldsymbol{x},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}det(\boldsymbol{\Sigma})^{1/2}}e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}(\boldsymbol{x}-\boldsymbol{\mu})} = \mathcal{N}(\boldsymbol{x},\boldsymbol{\mu},\boldsymbol{\sigma}) = \prod_{i=1}^{D}\frac{1}{\sigma_i\sqrt{2\pi}}e^{-\frac{\left(x_i-\mu_i\right)^2}{2\sigma_i^2}} \tag{3.9}$$

In the context of KDE's the *j^th^* kernel has its mean set to the data point, $\widehat{\boldsymbol{x}}^j$.

$$\mathcal{N}^j\left(\boldsymbol{x},\widehat{\boldsymbol{x}}^j,\boldsymbol{\sigma}\right) = \prod_{i=1}^{D}\frac{1}{\sigma_i\sqrt{2\pi}}e^{-\frac{\left(x_i-\widehat{x}_i^j\right)^2}{2\sigma_i^2}} = \prod_{i=1}^{D}\mathcal{N}_i^j\left(x_i,\widehat{x}_i^j,\sigma_i\right) \tag{3.10}$$

Using Eq. 3.8 and Eq. 3.10, the *D*-dimensional, normal, weighted KDE is formulated according to Eq. 3.11. Figure 3.2 shows a uniformly weighted KDE composed of three normal PD's.

$$p(\boldsymbol{x}) = \textstyle\sum_{j=1}^{N} w^j \mathcal{N}^j(\boldsymbol{x}, \hat{\boldsymbol{x}}^{\boldsymbol{j}}, \boldsymbol{\sigma}) \tag{3.11}$$

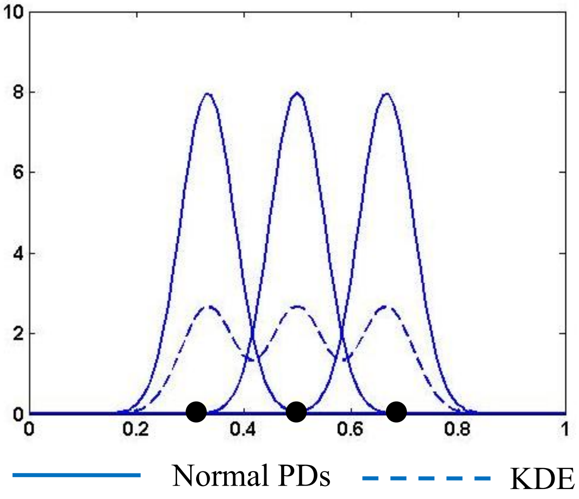


Figure 3.2: An Example Kernel Density Estimate

Using weighted KDE's for BN's requires calculating the conditional probability of Eq. 3.6 with Eq. 3.11. For the calculation of the *N*+1 value of the i^{th} variable, the joint probability of a variable's *K* parents with known values $\hat{x}_k^{N+1}$ can be expressed as Eq. 3.12.

$$p_i(\boldsymbol{pa}(x_i) = \hat{x}^{N+1}_{k=1\ldots K}) = \textstyle\sum_{j=1}^{N} w^j \prod_{k=1}^{K} \mathcal{N}_k^j\left(\hat{x}_k^{N+1}, \hat{x}_k^j, \sigma_k\right) \tag{3.12}$$

The conditional probability then becomes Eq. 3.13.

$$p_i(x_i|\boldsymbol{pa}(x_i) = \hat{x}^{N+1}_{k=1\ldots K}) = \frac{\sum_{j=1}^{N} w^j \mathcal{N}_i^j\left(x_i, \hat{x}_i^j, \sigma_i\right) \prod_{k=1}^{K} \mathcal{N}_k^j\left(\hat{x}_k^{N+1}, \hat{x}_k^j, \sigma_k\right)}{\sum_{r=1}^{N} w^r \prod_{s=1}^{K} \mathcal{N}_s^r\left(\hat{x}_s^{N+1}, \hat{x}_s^r, \sigma_s\right)} \tag{3.13}$$

The probabilities from the normal distributions of the parent variables have already been calculated and they can be collected into a coefficient, π, as shown in Eq. 3.14.

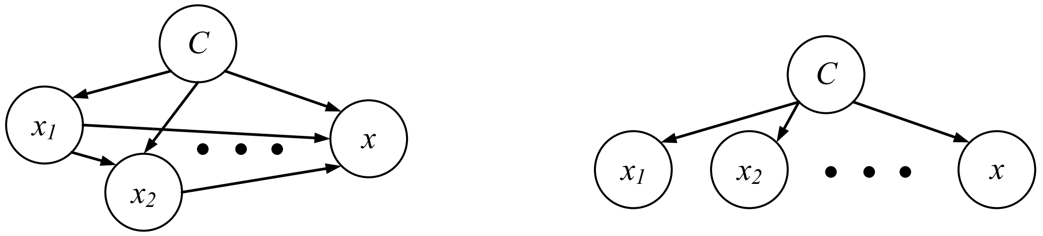
$$p_i\left(x_i|\boldsymbol{pa}(x_i)\right) = \sum_{j=1}^{N}\left(\frac{w^j \prod_{k=1}^{K} \mathcal{N}_k^j\left(\hat{x}_k^{N+1}, \hat{x}_k^j, \sigma_k\right)}{\sum_{r=1}^{N} w^r \prod_{s=1}^{K} \mathcal{N}_s^r\left(\hat{x}_s^{N+1}, \hat{x}_s^r, \sigma_s\right)}\right)\mathcal{N}_i^j\left(x_i, \hat{x}_i^j, \sigma_i\right) = \sum_{j=1}^{N} \pi^j \mathcal{N}_i^j \tag{3.14}$$

Hence, the PD of each variable is a weighted average of normal distributions with each weight, π^j, being a function of the parent probabilities and $\sum_{j=1}^{N} \pi^j = 1$.

The weights, w^j, can be used to determine the class conditional probabilities by uniformly weighting the design points that are known to be members of a class and setting the remaining weights to zero. This scheme requires storage of an array of weights for each class, c. Equation 3.15 shows the resulting class conditional formulation.

$$p_i(x_i|\boldsymbol{pa}(x_i), c) = \sum_{j=1}^{N}\left(\frac{w_c^j \prod_{k=1}^{K} \mathcal{N}_k^j\left(\hat{x}_k^{N+1}, \hat{x}_k^j, \sigma_k\right)}{\sum_{r=1}^{N} w^r \prod_{s=1}^{K} \mathcal{N}_s^r\left(\hat{x}_s^{N+1}, \hat{x}_s^r, \sigma_s\right)}\right)\mathcal{N}_i^j\left(x_i, \hat{x}_i^j, \sigma_i\right) = \sum_{j=1}^{N} \pi_c^j \mathcal{N}_i^j \tag{3.15}$$

Two extreme cases of network connectivity represent very well studied classifiers: Naïve Bayes for the fully independent BN given the class (John and Langley, 1995), and Parzen windows for the fully dependent case (Jain and Ramaswami, 1988). Figure 3.3 depicts the graphical representation of these two methods for the purposes of classification where the dependency on class is represented by the root C node of the graph.



$$p(\boldsymbol{x}|c) = p(x_D|x_{D-1} \dots, x_1, c) \dots p(x_2|x_1, c)p(x_1|c)$$

$$p(\boldsymbol{x}|c) = p(x_D|c) \dots p(x_2|c)p(x_1|c)$$

Figure 3.3: Parzen Window (left) and Naïve Bayes Classifiers (right).

Equations 3.3-3.5, 3.15 define the kernel-based Bayesian network (KBN) classifier (Perez et al., 2009). These foundations were implemented in Matlab® as a collection of m-files that are presented in Appendix A. As discussed in the previous section, this classifier is used for two purposes: to classify a new point and to sample the acceptable region. The procedures for performing these tasks and the associated time complexity are described next.

3.3 EVALUATING, SAMPLING, AND TRAINING

This section presents the time complexity for the three primary uses of the proposed classifier: training, classifying a new point, and sampling one of the class conditional probability distributions. One of the strengths of using the BN representation of the class conditional probability distributions is the ability to trade off the computational overhead of classifying and sampling for representational capability. The time complexity for these tasks is determined by the longest chain of dependencies in the BN from the root to the leaf nodes. As the longest chain of dependencies is reduced and hence the time complexity is reduced, the representational capability of the classifier is also reduced, providing the capability to scale the computational overhead to the difficulty of the problem.

Both evaluation and sampling of the classifier proceeds in ancestral order: beginning with evaluating or sampling the distributions of the root nodes and proceeding to their children's distributions, and then to their children's children nodes, and so on until the leaf nodes have been evaluated or sampled. Thus the longest chain of dependencies, of length L, is the outer loop that determines the time complexity of both evaluating and sampling the flexible classifier (assuming a parallel implementation). For the fully connected graph this length is equal to the dimensionality of the problem, *D*. For the fully disconnected graph $L = 1$. At each node in the graph, the conditional probability distribution of Eq. 3.15 has to be either evaluated or sampled.

In order to classify a new point, the PD of both the acceptable and unacceptable regions is evaluated. This involves an inner loop over all *N* training points. Thus the time complexity of classifying a new point is $\mathcal{O}(NL)$. In order to sample the PD of Eq. 3.15, three steps have to be taken: 1) the *K* mixing coefficients, π, of the parent nodes have to be calculated, 2) the resulting discrete multinomial probability distribution of *K-1* degrees of freedom needs to be sampled, and 3) the 1 dimensional normal distribution determined from step 2 needs to be sampled. The combined time complexity of these 3 steps will be summarized as $\mathcal{O}(K)$. The time complexity of sampling the BN would hence be $\mathcal{O}(NLK)$ except for the possibility that some samples will lie outside of the desired maximum and minimum limits for the design parameters. Samples lying outside of the design space domain will have to be rejected. If this occurs *M* times, the resulting time complexity of sampling the BN is $\mathcal{O}(NLKM)$. As the number of training points increases, the width of each kernel PD can be decreased which in turn will decrease *M*.

Furthermore, the acceptable region of the design space may be far enough from the search domain boundary such that the probability of rejection is essentially zero. Thus the time complexity of sampling will most likely be $\mathcal{O}(NLK)$. The time complexities of using the flexible classifier are summarized in Table 3.1, including the training time complexity. It should be noted that another optimization is possible that performs all or a fraction of the N calculations of each training point's contribution to the PD in parallel. However, to fully realize this optimization would probably require a high number of parallel processors.

Table 3.1: Time Complexity for using the KBN Classifier

	Time Complexity		
Task	**Fully Disconnected**	**General**	**Fully Connected**
Training	1	1	1
Classifying	$\mathcal{O}(N)$	$\mathcal{O}(NL)$, $1 \leq L \leq D$	$\mathcal{O}(ND)$
Sampling	$\mathcal{O}(NK)$	$\mathcal{O}(NKL)$, $1 \leq L \leq D$	$\mathcal{O}(NKD)$

The time to train the classifier is minimal because all that is required is to store the training points in an array. However, a training time complexity of one assumes that the correct classification is known for the training points. This research follows the assumption that either a simulation or an experiment can be performed to determine the "correct" classification. Hence the cost of "correctly" evaluating the acceptability of the training points is equal to the cost of running N simulations or experiments. However, it is common throughout the literature to assume that the class of each training point is known at training time. In addition to assuming that the correct classification is known, the low cost of training also assumes that all other algorithm settings are preset through an easily evaluated rule such as a formula or table. This need not be the case. The KBN classifier has several inputs that need to be either set by the user interactively, preset according to some rule, or adaptively set through some predefined procedure. These options are discussed in the next section.

3.4 THE KBN CLASSIFIER'S SETTINGS

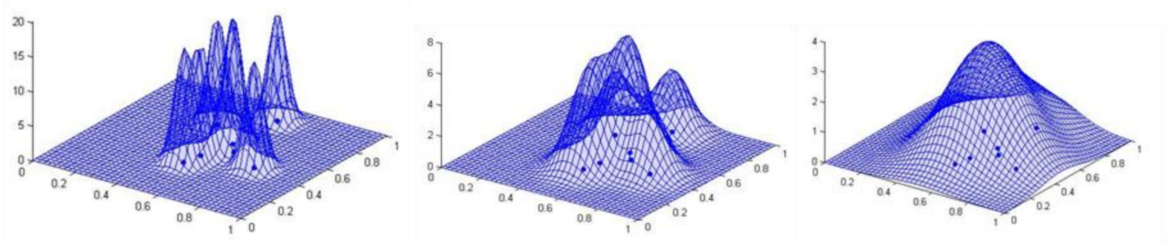
The user of the KBN classifier has several decisions to make: the graph connectivity, the kernel widths (standard deviations), the loss factors, the search domain, and the training points all have to be determined either in advance of training the classifier or as an augmentation to the training procedure. The training data includes both the design parameters as well as the classification for each design point in terms of being either acceptable or not. In the next chapter, an example unmanned aerial vehicle (UAV) design problem is presented that is used throughout this research as a motivating example for exploring methods that help decide which standard deviations and training data to use. The use of loss factors is presented once in Chapter 4 as an illustration of their potential use to a designer. The choice of graph connectivity is only lightly touched upon in Chapter 4, leaving an in depth treatment of this topic to future work. The maximum and minimum values for the design parameters that define the search domain have been assumed to be either available from expert knowledge or preliminary experimentation. The important choice of how to set the standard deviations will now be discussed in more depth.

Research has shown that the choice of smoothing parameter or, for the case of Gaussian kernels, the standard deviation, is the most important consideration when using KDE's (Scott, 1992). Small standard deviations result in very peaked distributions with a distinct maximum centered on each data point, while large standard deviations smooth out the influence of each data point into a single peaked distribution as shown in Fig. 3.4. In between these extremes the distribution will allow for both local structure as well as good generalization results to interpolated points. Two methods for setting the standard deviation are used in this research. The first approach follows (John and Langley, 1995) in using Eq. 3.16 as a heuristic for setting the kernel width for the classifiers with an additional scale factor, α_c. Note that Eq. 3.16 with $\alpha_c = 1$ was the formula used in (John and Langley, 1995).

$$\sigma_c = \frac{\alpha_c}{\sqrt{N}} \tag{3.16}$$

This research also follows the practice of normalizing the data according to Eq. 3.17 such that all design points are between zero and one for each dimension (Perez et al., 2009; Siminoff, 1996). The normalization is provided by the search domain limits on each dimension as defined by minimum, $x_{i,min}$, and maximum, $x_{i,max}$, allowable values for each dimension. Normalization allows one to consider one standard deviation per class.

$$x_i = \frac{\hat{x}_i - x_{i,min}}{(x_{i,max} - x_{i,min})} \tag{17}$$



$\alpha = 0.1$ $\alpha = 0.2$ $\alpha = 0.4$

Figure 3.4: The Effect of Standard Deviation on the Kernel Density Estimate

The strength of using Eq. 3.16 to set the standard deviations is its fast evaluation time. However, experimentation is required to set the scale factors, α_c, for each class. Furthermore, the rigid functional form might not reliably lead to decision boundaries that always correctly classify all of the training data.

The second approach to setting the standard deviations adaptively finds the largest standard deviation of each class that still correctly classifies all of the training points to a specified posterior class probability. Adaptive approaches will have longer training times, but they do have the potential to avoid the experimentation required to find a good rule. In addition, the novel method for adaptively setting the standard deviations developed in Chapter 5 was designed to generate decision boundaries that are reliably consistent with a designer's knowledge at the time of training.

Observe that in Eq. 3.18 the standard deviations decrease as the number of training points increases, resulting in sharper kernels. Sharper kernels are a result of the higher density of training points having an increased capability to capture local details in the classified regions of the design space and hence being able to reduce the classification error. The next section discusses this relationship in greater depth.

3.5 REPRESENTATION CAPABILITY

A classifier's performance can be measured by the percentage of test points that are misclassified, called the error rate. The more training points there are, the more kernels that are available to approximate the decision boundary, ideally leading to lower error rates. If the classification error can be reduced by increasing the number of training points, can the error rate be made arbitrarily small for any shape of the acceptable region? Are there theoretical restrictions to the shapes of regions that can be represented by the KBN classifier given an unlimited number of training points? In this section we show the relationship between Parzen window classifiers and radial basis function (RBF) networks following (Web, 2002). RBF networks have the property of being able to approximate with arbitrary accuracy any continuous function given that the RBF network is of the form of Eq. 3.19 with a kernel function, *K*, that is bounded, continuous, and not a polynomial (Liao, et al., 2003). Here y is used in place of p from previous equations to emphasize that the result need not be a probability distribution.

$$y = \sum_{i=1}^{N} w^i K\left(\frac{x-\boldsymbol{\mu}^i}{\sigma}\right) \tag{3.19}$$

By defining the weights, w^i, according to Eq. 3.20 and setting the centers, $\boldsymbol{\mu}^i$, of the kernel function to the sample points, $\boldsymbol{x}^i$, the Parzen window classifier with Gaussian kernels has a posterior probability of the class given the design point, repeated here as Eq. 3.21, with the same functional form as the radial basis function network.

$$w^i = \frac{P(c)}{p(\boldsymbol{x})N} \tag{3.20}$$

$$p = \sum_{i=1}^{N} \frac{P(c)}{p(\boldsymbol{x})N} \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{x}^i, \sigma\right) \tag{3.21}$$

Eq. 3.21 meets all of the requirements for being a universal approximation to continuous functions, however, the weights of Eq. 3.20 for the fully connected KBN classifier are not solved for directly as they are in RBF networks, and the centers of the kernels are restricted to the finite number of sample points. Thus the representational capability of the fully connected KBN classifier used in this research is limited. In practice, the user of the KBN classifier will have to live with a certain amount of classification error that depends upon the number of samples as well as the complexity of the shape of the classified region. The error due to the shape of the classified region being beyond the representational capabilities of the classifier is not reducible. The error due to having a finite number of samples is reducible by taking more samples, although there are practical limits to this remedy.

While the fully connected KBN classifiers are not universal approximators, they are capable of representing any probability distribution arbitrarily accurately as the number of training points increases, a property called strong point-wise consistency (Scott, 1992). In fact, (John and Langley, 1991; Perez et al., 2009) have shown that as long as the BN connectivity correctly represents the true conditional independencies in the underlying probability distribution, all of the KBN classifiers are strongly point-wise consistent. However, the feasible regions are not likely to be generated from probability distributions. The feasible regions will typically come from inequality constraints on the output parameters of nonlinear simulations and hence have no connection to being represented by probability distributions.

The properties of universal approximation and strong point-wise consistency both suggest that the classification error should decrease with more training points. However, the more training points that are required, the higher the initial development cost of the classifier and the higher the cost of classifying new designs. For difficult problems of a high dimensionality, a large number of training points might be required to achieve an acceptable level of misclassification error. For these difficult problems either more resourceful classifiers must be used or the correct answer will be determined directly through a simulation or experiment.

For these reasons, the KBN classifier used in this research cannot be guaranteed to meet the requirements of the hypothesis that arbitrarily shaped regions of the design space can be represented. Nevertheless, the KBN classifier will be demonstrated in the following chapters to have better representational capability than the simpler interval-based classification. Furthermore, kernel density estimation has known advantages to the use of histograms. With KDE's, the influence of each design point is with respect to its distance to other design points as opposed to its presence in a predesigned grid. The resulting probability distribution is also smooth and hence more easily searched—a feature that will be exploited for the adaptive sampling developed in Chapter 7. Furthermore, classifiers take advantage of more knowledge than histogram-based methods because they use both the feasible and infeasible design points to determine the decision surface. For all of these reasons, the KBN classifier is a good candidate for advancing the state of the art in set-based collaborative design. While methods such as RBF networks exist that are universal approximators, they come at a higher computational cost than the proposed method because of the matrix inversion used to solve for the weights. Future work needs to explore the tradeoff between increasing computational cost and more flexible classifiers. The extent to which universal approximation is important enough to justify the additional computational cost needs to be better understood.

3.6 DISCUSSION

KBN classifiers were presented in this chapter for the stated purposes of this research: to classify and sample arbitrarily shaped and potentially disconnected acceptable regions of the design space. The connectivity of the Bayesian network can be used to control the complexity of the classifier model, trading off representation capability with computational cost of classifying or sampling. In its most complex form, the KBN classifier almost meets the ambitious representational requirements for this research: approximating arbitrarily shaped and potentially disconnected design regions with asymptotically reducing error as a function of the number of training points. Despite this theoretical shortfall, the representational capability of the classifier is rich enough to demonstrate acceptably low classification error rates with a reasonable number of training points for the example collaborative design problem presented in Chapter 4. The KBN classifier will also be demonstrated to perform significantly better than the interval-based classification most commonly used for set-based collaborative design. Furthermore, because the flexible classifier uses probability distributions to represent the classified regions it facilitates the adaptive sampling methods developed in Chapter 7 because the mutually acceptable region of the design space can be sampled.

Classifiers other than the KBN classifier presented in this chapter could be used for all of the purposes of this research. Classifiers, such as Bayesian networks with mixtures of Gaussians (Davies and Moore, 2000) and relevance vector machines (Tipping, 2001), can provide faster evaluation times at the expense of increased training time while maintaining the representation and sampling capabilities of the KBN classifier. Classifiers that can be sampled are called generative. Classifiers that are not generative but are determinant, such as RBF networks (Powell, 1987) and support vector machines (Vapnik, Golowich and Smola, 1997), do not produce probability distributions over the design space that can be sampled. Determinant classifiers can be used for all of the purposes of this research other than the adaptive sampling presented in Chapter 7.

The user of the KBN classifier must choose how to set several tuning parameters, most importantly the smoothing parameters. Some of these choices are elaborated upon in the following chapters of this dissertation with Chapter 5 being dedicated to choosing the smoothing parameters and Chapter 7 being dedicated to choosing the training points. Choosing the graph connectivity and the loss factors are only briefly discussed in Chapter

4, while choosing the search domain is not discussed any further. More detailed treatments of choosing the graph connectivity, the loss factors, and the search domain have been reserved for future work. Chapter 6 presents the possibility of incorporating expert knowledge in the form of monotonic relationships to improve the efficiency of the KBN classifier.

Once a designer has constructed the classifier of their design space, choices remain for how to share the classifier with their collaborating designers as well as how to incorporate the results of other designer's classifiers into one's own classifier. These coordination decisions are elaborated upon in depth in the next chapter where it is shown how the classified acceptable region of a design space can be mapped through simulations onto another group's design space. Chapter 8 extends the results of Chapter 4 to the case of sharing classifiers between groups with common design variables.

Chapter 4. Classifiers for Collaborative Design: Vertical Coupling

This chapter has four purposes. First, the KBN classifier just developed in Chapter 3 is demonstrated which should help solidify the concept of locally using classifiers to represent the feasible regions of a design space. Second, this chapter demonstrates how classifiers can be used collaboratively to map one designer's feasible region into another designer's design space—an example of vertical coupling in hierarchical collaborative design. Third, the simplified UAV wing collaborative design problem is introduced and discussed as an ongoing demonstration problem used throughout this dissertation to illustrate the proposed methods. Finally, the enhanced representational capability of the KBN classifier with respect to interval classification is demonstrated. In summary, this chapter will demonstrate the usefulness of solving collaborative design problems with classifiers that have a richer representational capability than intervals. The UAV wing design demonstration application's relevant features will now be introduced, leaving the details of the problem formulation to Appendix B.

4.1 THE UAV DEMONSTRATION PROBLEM

Consider the problem of designing an unmanned aerial vehicle (UAV) wing for the cruise condition, where the goal is to achieve a range greater than 900 kilometers. This problem has been decomposed according to Fig. 4.1 where the aerodynamics team is responsible for determining the wing loads based upon the external wing geometry, the structures team is responsible for determining the wing weight for a structurally sound wing based upon the external and internal geometry, and the systems team determines the UAV range based on the results of the aerodynamics and structures teams. The airfoil geometry has been parameterized according to the NACA four-digit, three-parameter

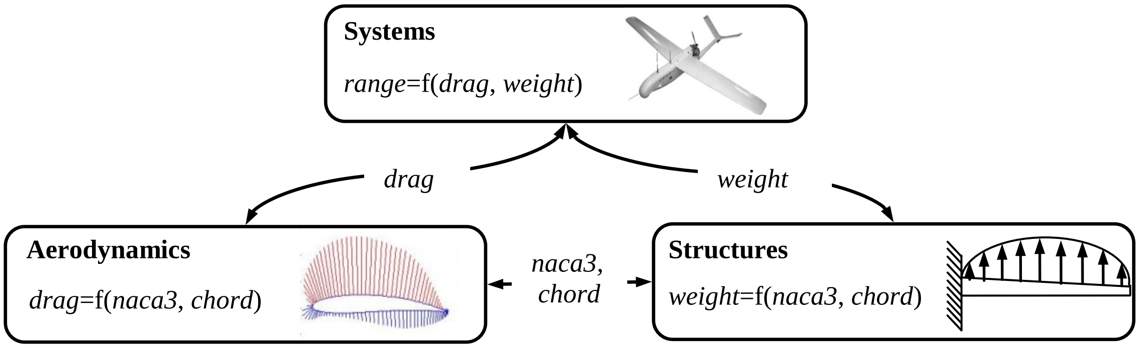


Figure 4.1: A Collaborative Design Problem for a UAV Wing

standard using *naca1*, *naca2*, and *naca3* as the design variables (Abott and von Doenhoff, 1959). In addition to these airfoil parameters, the rectangular wing's external geometry is also characterized by the wing *chord*, *span* and *angle of attack*. The structures and aerodynamics teams have these variables in common. We will consider further simplifications that result in an easily visualized design space by fixing the first two NACA parameters which define the airfoil camber, to the reasonable values of 4 and 5 respectively. We also fix the *angle of attack* to 0.6 degrees which is near optimal for this airfoil. The *span* is automatically adjusted to ensure there is enough planform area to achieve a lift of 35 N during cruise. The remaining design variables common to both the aerodynamics and the structure's design problems are *chord*, which is the front to back width of the wing, and *naca3*, which is the wing's top to bottom thickness normalized by the chord.

This example problem is representative of the hierarchical decomposition of complex engineering products: the results of subsystem simulations feed into the simulation of the product's performance at the system level. In addition to this vertical coupling across scales, the subsystems share common design parameters that must be in agreement by the end of the design process. Each design team is responsible for their analysis, a responsibility that should not be simply handed over to other teams who do not have the expertise required to validate the results. The analyses have been simplified for the sake of research but still maintain enough fidelity such that they correctly capture the trends between the inputs and the outputs of each group.

In low Reynolds number cruise speeds, the *drag* is dominated by induced drag as determined by the aspect ratio (the *span* divided by the *chord*) which is controlled in this case by the *chord*. The lower the *chord*, the higher the *span* needs to be in order to create enough lift. A lower chord and higher span increases the aspect ratio and hence reduces the induced drag. However, high aspect ratio wings are long and slender and require more internal material in order to keep them structurally sound, and hence they produce higher weight designs. Furthermore, the *naca3* parameter is the wing thickness normalized by the chord, and therefore a higher chord also allows for a higher thickness wing. Having a thicker cross-section can lead to a more efficient distribution of material for increasing the wing's strength. Hence, the fundamental conflict of interest for this problem is that the aerodynamics team seeks to minimize *drag* by decreasing the *chord*, and the structures team seeks to minimize *weight* by increasing *chord*. Their conflict can be resolved at the systems level where the resulting *range* of the UAV based upon the subsystem results of wing *drag* and wing *weight* can be used to determine the best tradeoff between *weight* and *drag*.

Low fidelity and fast physics-based design programs were written to capture these effects. For aerodynamics' calculation of lift and *drag*, a linear vortex panel method and a boundary layer growth calculation were created according to (Katz and Plotkin, 2002) and (Moran, 1984), respectively. Structure's calculation of weight depends on adding enough skin thickness to the wing so that the wing doesn't yield under 10X static cruise loads based upon an arbitrary cross-section beam bending formula from (Cook and Young, 1999). The systems' calculation uses the Breguet range equation (Raymer, 2006), once the fuselage aspect ratio has been optimized for maximum fuel capacity and minimum drag using an empirical drag equation from (Hoerner, 1965). These details are noteworthy not just for being physics-based but also because they expose the details of each group's design problem which include internal iterations and optimizations that must be repeated for every new input. Most importantly, if the subsystem design teams were given simple directives to minimize drag and weight then they would not converge to the same design; coordination is necessary. The coordination strategy that we follow next uses KBN classifiers to map and share feasible regions of the design space. The use of interval-based classification of the design space is also presented to demonstrate the improved representation capabilities of the proposed method relative to the simpler and more common alternative. The Matlab® code used to produce the following results is presented in Appendix C.

4.2 UAV SOLUTION 1: BASELINE WITH VERTICAL COUPLING ONLY

The first design process presented here is simple enough to demonstrate the most elementary of applications of KBN classifiers to collaborative design. The first simplification is the coordination of the choice of design points for the shared parameters of the subsystems design teams such that for every point of *naca3* and *chord* aerodynamics and structures will calculate a *drag* and *weight* respectively. Thus the two subsystem teams can be merged into a single team that agrees on how the values of their shared parameters are sampled. The resulting problem decomposition is purely vertical as shown in Figure 4.2. The second simplification is a sequential design process where the systems group first explores and classifies their design space before the subsystems

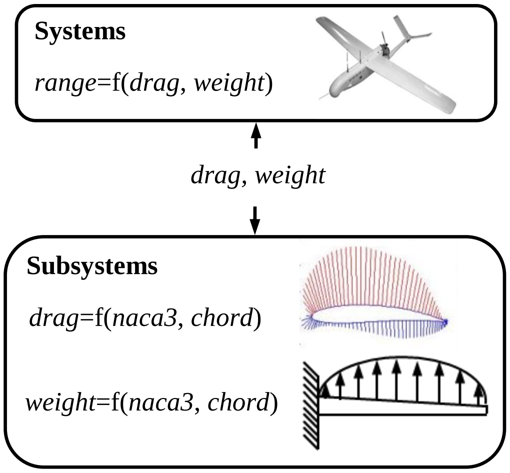


Figure 4.2. The UAV Design Problem with Merged Subsystems Teams

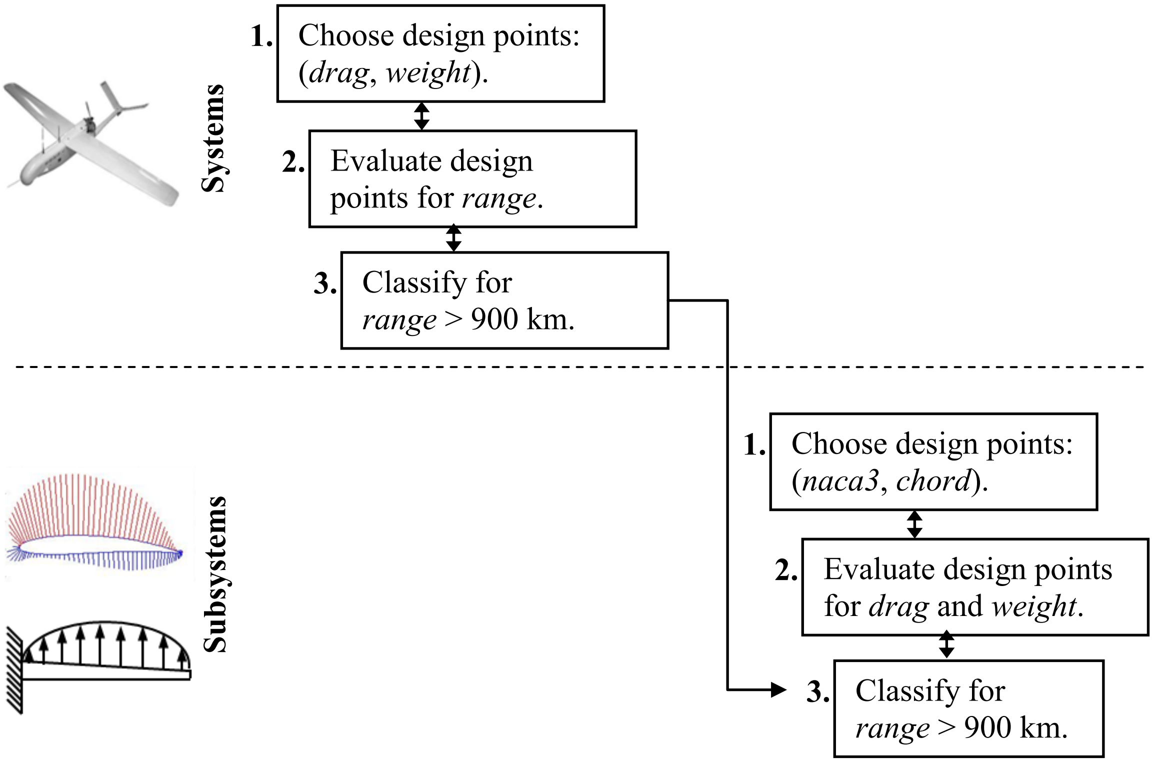

Systems
1. Choose design points: (drag, weight).
2. Evaluate design points for range.
3. Classify for range > 900 km.
Subsystems
1. Choose design points: (naca3, chord).
2. Evaluate design points for drag and weight.
3. Classify for range > 900 km.

Figure 4.3. The UAV Collaborative Design Process

group classifies their space. The process flow is depicted in Fig. 4.3. For this initial design process, each group is free to choose and evaluate their design points within their chosen design space domains independently. The only aspect of the design process of Fig. 4.3 that is sequential is that the classification of the subsystem design space will use a thoroughly developed classifier at the systems level in order to map acceptable regions of the system level design space in terms of *drag* and *weight* to acceptable regions of the subsystem level design space in terms of *naca3* and *chord*. This section presents how classifiers that are well developed (i.e. have low error rates) can be propagated vertically through each group's nonlinear simulations starting at the highest level requirement for a range greater than or equal to 900 km. Alternative design processes are discussed after this baseline design process has been demonstrated.

The systems group's design process begins with a Halton sequence (Halton, 1960) of 100 design points. This choice of sampling was used for its space filling yet irregular pattern as well as its ability to be sampled sequentially with a progressively finer sampling resolution without the need to know how many sample points to use beforehand. These points are the training samples for the classifiers. At the systems level, the design points are evaluated for a range and classified according to whether or not the range is greater than or equal to 900 km. The combined aerodynamics and structure's team also explores their design space using a Halton sequence of 100 design points. The only choice in construction of the Halton sequence is the intervals for the design parameters. Once minimum and maximum values have been chosen for each design parameter, the sequence of design points is fully determined. It is assumed that enough prior knowledge is available to choose effective intervals for the design space domain. Table 4.1 presents the maximum and minimum design parameter values used in this example. In addition to the design intervals, each design group must choose the standard deviation of the Gaussian kernels. Equation 3.16 is used as the rule for setting the standard deviations for this solution. The scale factor, α_c, is chosen through experimentation as if the designer were interactively editing the resulting classifiers. Table 4.2 presents the values of α_c that are used for each design group's classes.

Table 4.1: UAV Design Parameter Limits

Parameter	**Minimum Value**	**Maximum Value**
drag	0	1
weight	0	5
naca3	1	15
chord	0.1	0.4

Table 4.2: Scale Factors for Rule-Based Calculation of the Kernel Widths

	Acceptable Class	**Unacceptable Class**
system	0.25	0.25
subsystem	0.10	0.50

The remaining settings for a completely determined classifier are the loss factors and the BN connectivity. In this section we present solutions for both the fully connected BN, the Parzen window classifier, and the completely disconnected BN, the naïve Bayes classifier, at both the systems and subsystems levels. Initially, the loss factors are chosen to not have an influence on the decision. The effect of the loss factors will be presented in the next section. The results from an interval classifier are also presented in this section. Before showing these results, an example is presented in Fig. 4.4 of how the systems level probability distributions and decision boundary change to capture the correct decision boundary for 10, 20 and 100 training points for the fully connected KBN classifier without loss factors. The correct contour, based upon calculating the ranges for a grid of 10,000 points and using the contour function in Matlab®, is also shown as a solid line in the 100 training point graph at the bottom left of Fig 4.4.

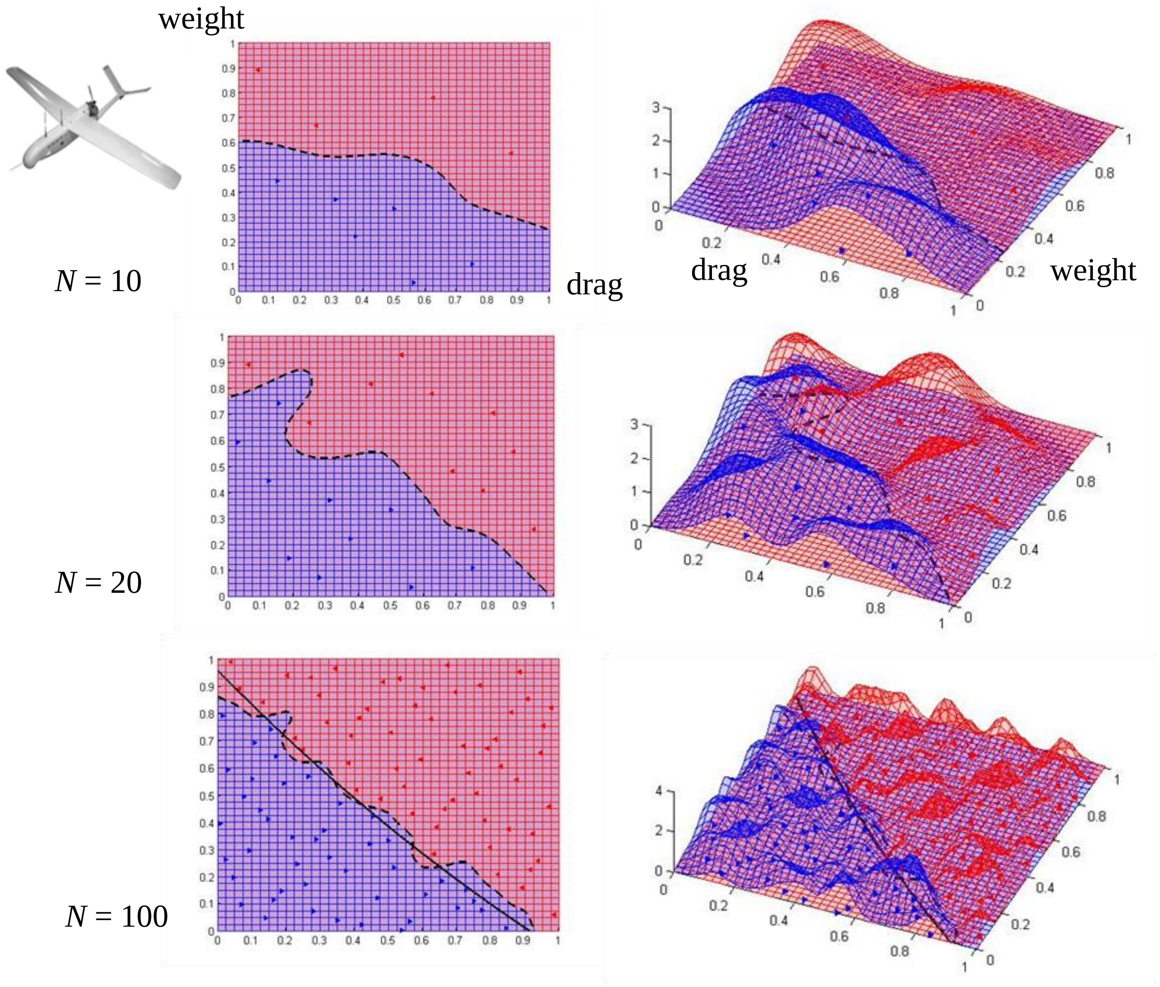


Figure 4.4. The Systems' Classifier for 10, 20, and 100 Training Points

Figure 4.5 presents the decision boundaries of the system level fully connected, fully disconnected, and interval classifiers after all 100 samples have been evaluated. At

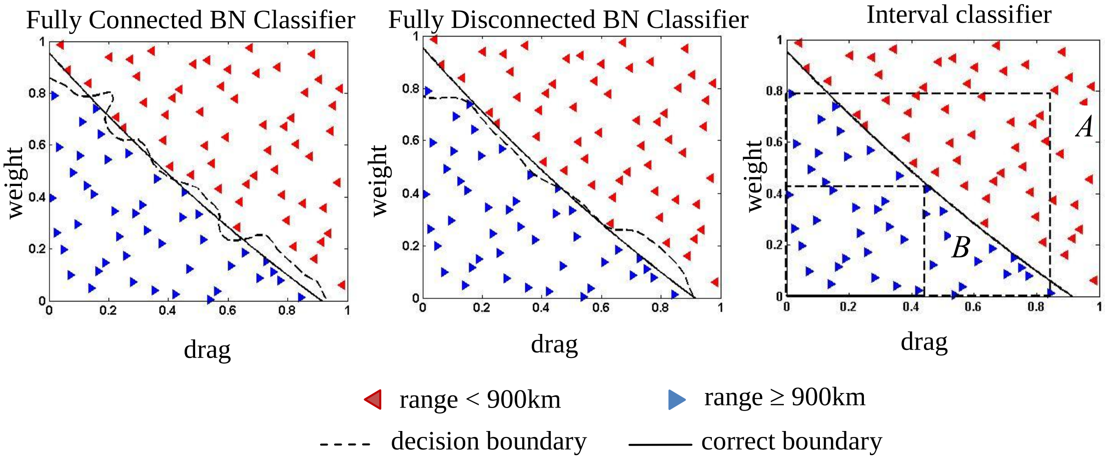


Figure 4.5. The Classifier's Affect on the Systems' Decision Boundary

each stage of sampling and classification, a set of 1000 test samples generated as a Hammersley sequence (Hammersley, 1960) was used to find the fraction of the designs that are classified as having ranges greater than or equal to 900 km but that actually have ranges less than 900 km, called the false positive error rate. Likewise, the false negative error rate can be found as the fraction of designs that are classified as having ranges less than 900 km but that actually have ranges greater than or equal to 900 km. These error rates are shown in Fig. 4.6 as a function of the number of training points. The significant result is that the error rates decay to near zero for both the naïve Bayes and the Parzen window classifiers. The fact that the naïve Bayes classifier can achieve near zero error rates is perhaps surprising given the assumption of independence between the variables. However, as can be seen in Fig. 4.5, the assumption of independence between variables does not mean that the decision boundary must be parallel to the variable axes. Considered independently, the density estimates will be higher for both parameters at the lower ends of their domains. Multiplying two such distributions together will result in higher probability estimates in one corner that diminish with distance from that corner, producing the diagonal decision surface in the center of Fig. 4.5.

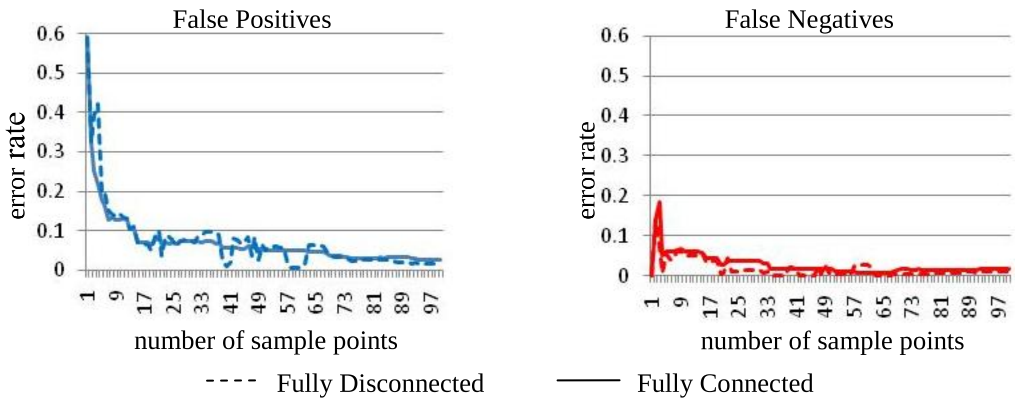


Figure 4.6. KBN Classifier Error Rates for the Systems' Design Space

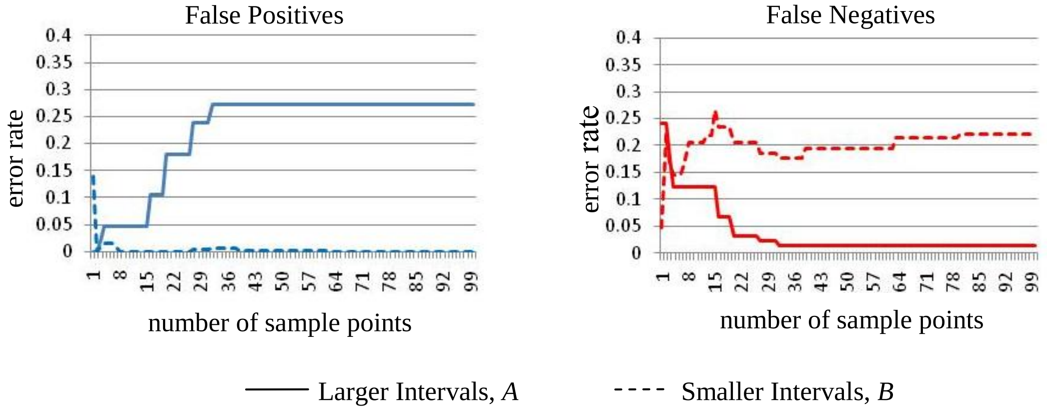


Figure 4.7. Interval Classifier Error Rates for the Systems' Design Space

Using intervals as classifiers for each design parameter presents a challenge when classifying the systems design space. At one extreme, the intervals over the *drag* and *weight* can be chosen to correctly classify all of the known good designs. This strategy produces larger intervals, labeled A in Fig. 4.5, that reduce the false negative error rate but increase the false positive error rate. At the other extreme, small enough intervals, labeled B in Fig. 4.5, can be chosen that eliminate all false positive error rates but at the expense of allowing more false negative error rates. This fundamental tradeoff in interval-based classification schemes is shown in Fig. 4.5 for the case of all 100 training samples. The error rates are shown in Fig. 4.7 as a function of the number of training points. The significant result is that, for this classification problem, the error rates from using intervals are high and do not converge with larger numbers of sample points. The error rates for interval classification have converged to a fixed total of about 25% after 100 samples. In contrast, the error rates for the BN classifiers have converged to a total of less than 5% after 100 samples.

Following the process flow chart of Fig. 4.3, the subsystem classification process can proceed now that the systems classifier exists with sufficient detail. For this solution, the design points in terms of *naca3* and *chord* for aerodynamics and structures have been coordinated to be equal for the two subsystems, such as might be done in a planned experiment. In this case, for every subsystem design point from the Halton sequence, both the *weight* and the *drag* are calculated by subsystem models; so, they are known at the time of classification. Thus, the system level KBN classifier can be used directly to determine if a given subsystem design point's combination of *weight* and *drag* will result in a range greater than or equal to 900 km, without running the design point through the systems' simulation. If either the *weight* or the *drag* is outside of the system level design space, the design point is classified as unsatisfactory. In this manner, the subsystem teams map the system level classification to a classification over their design variables (i.e., naca3 and chord).

From Fig. 4.8-4.11 we see that the subsystem level KBN classification produces an island for which the classification error rates decrease as more points are sampled.

The correct decision boundaries are also shown as a contour determined by a 10,000 point grid of design points whose actual ranges were calculated. For the interval

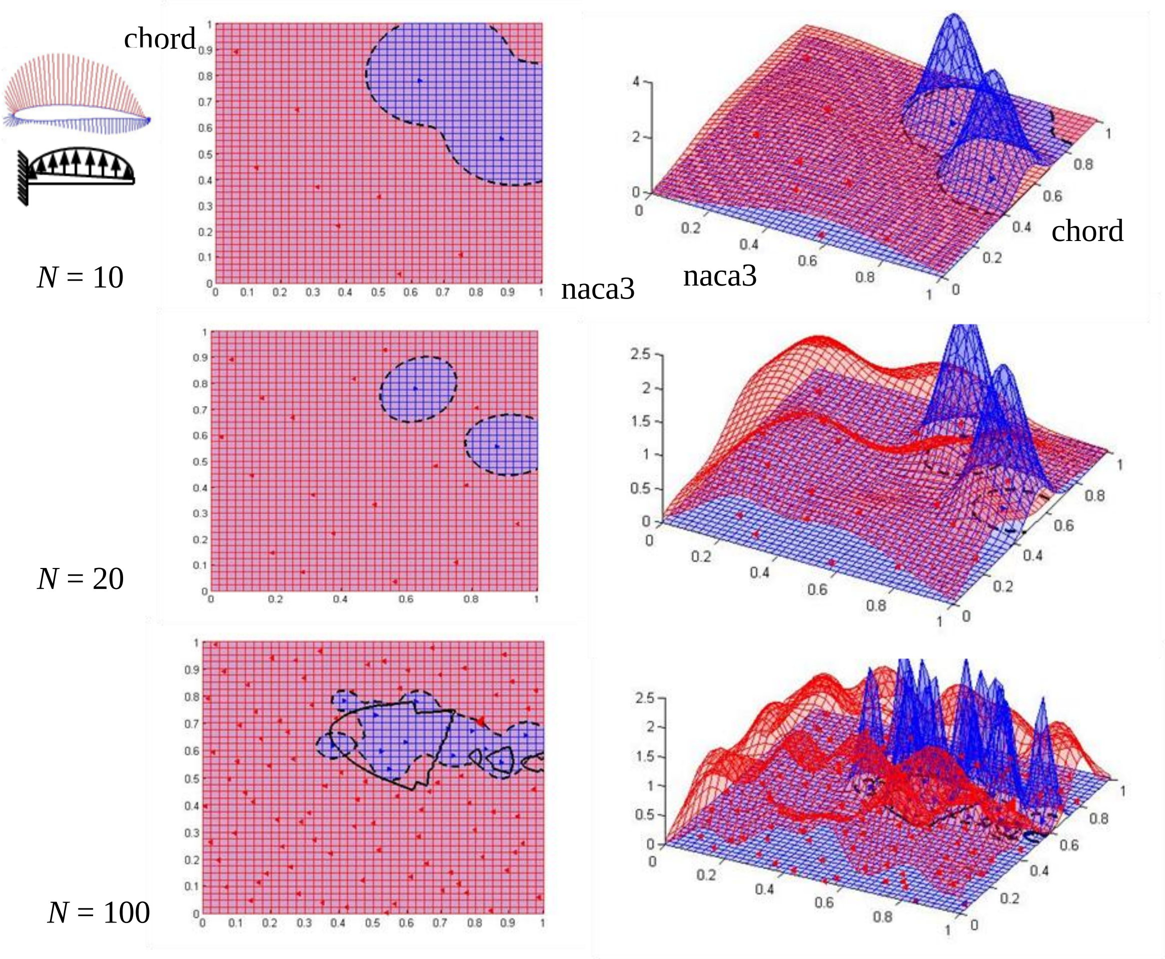


Figure 4.8. The Subsystems' Classifier for 10, 20, and 100 Training Points

classifiers, subsystem design points were classified using a system-level interval chosen to be half way between intervals A and B of Fig. 4.5 because the smaller interval B classified zero satisfactory designs at the subsystem level. Again, for the subsystem classification problem, the interval error, shown in Fig. 4.11 converged to a high fixed rate of about 17% after 100 samples. In contrast, the KBN error rates decreased to less than 10% after 100 samples.

These results demonstrate the improved ability of KBN classifiers to map the satisfactory regions of the design space relative to interval-based approaches for the

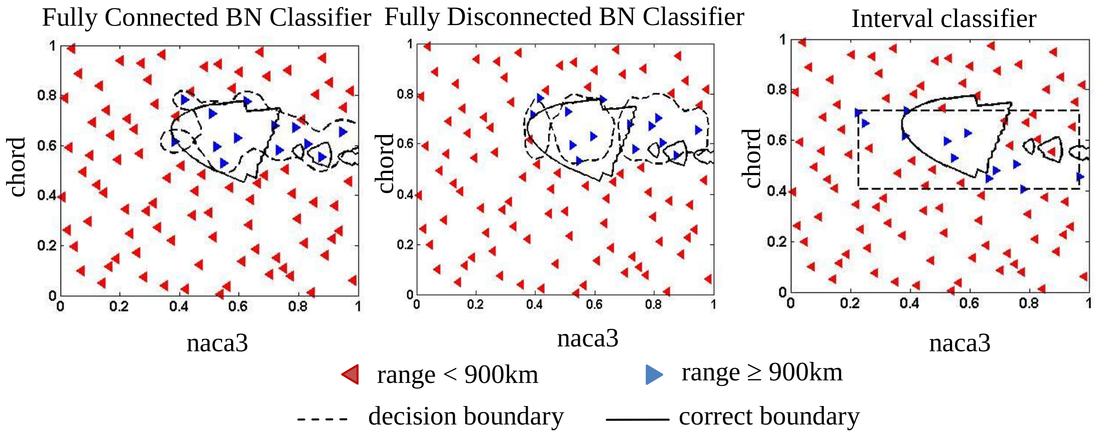


Figure 4.9. The Classifier’s Affect on the Subsystems’ Decision Boundary

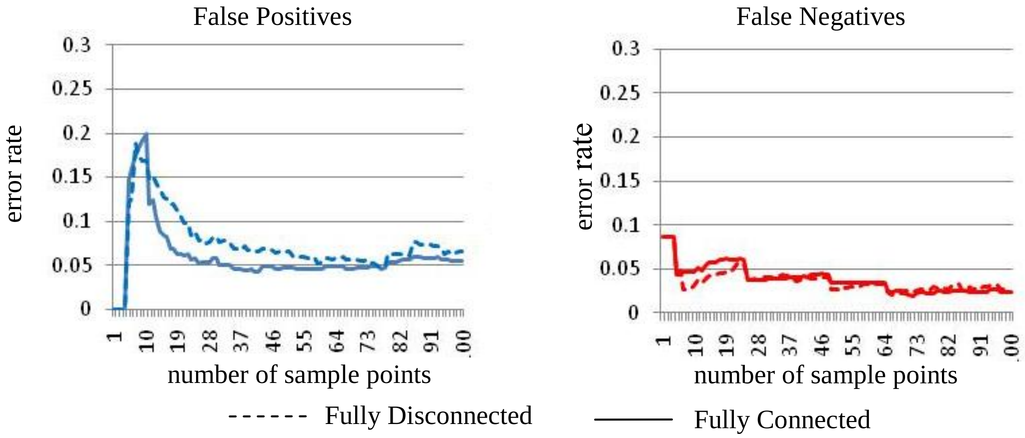


Figure 4.10. KBN Classifier Error Rates for the Subsystems’ Design Space

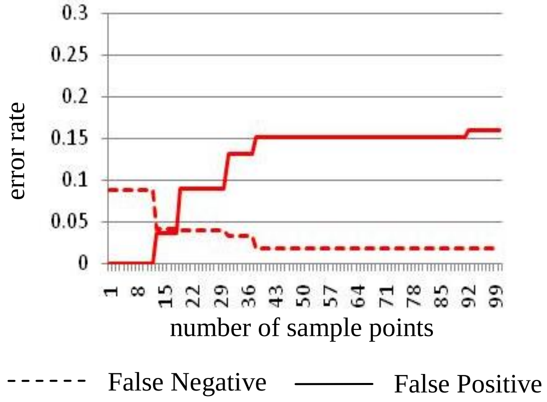


Figure 4.11. Interval Classifier Error Rates for the Subsystems' Design Space

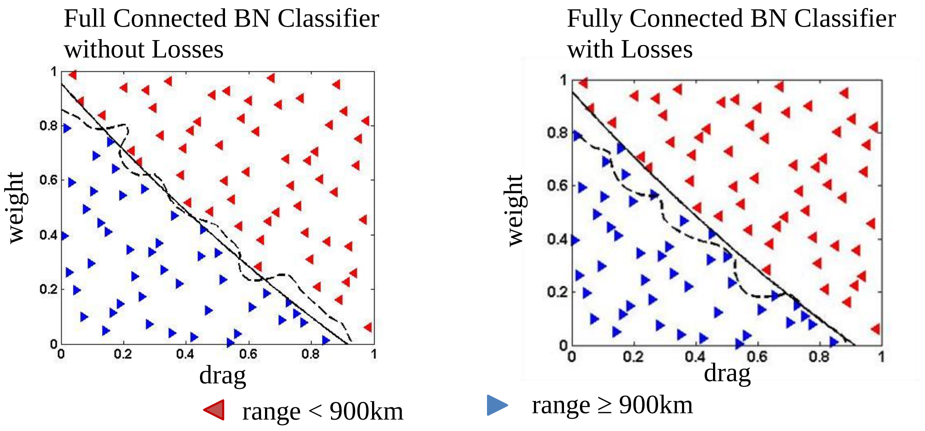
presented example problem. This improvement is primarily a result of the ability of KBN classifiers to produce decision boundaries that are not necessarily parallel to the parameter axes. The error rates of interval classifiers are compounded by the nonlinear mapping from the subsystem design spaces to the system level calculation of the UAV range. If the correct UAV range for each of the subsystem design points were calculated and used to classify the subsystem design space directly, the subsystem intervals would produce lower error rates. These results show that satisfactory regions do exist for which interval-based classification errors will always be relatively high no matter how complete our knowledge of the design space; they are a fundamentally limited representation of our knowledge. For these types of spaces, KBN classifiers can perform better than interval methods, even with the fully disconnected BN classifier's assumption of independence between parameters. Additionally, KBN classifiers appear to preserve the ability to converge to low error rates even when they are mapped to another design space, although more test problems are needed to verify this result. The next example presents how loss factors can be used to trade off false positive and false negative error rates for the case when a misclassification is too costly.

4.3 UAV Solution 2: Baseline Plus Loss Factors

The error rates in KBN classifiers can be traded off using the loss factors in a manner similar to how false positive and false negative error rates were traded off with larger or smaller intervals. The loss factors allow us to shift the decision boundary inward or outward, trading off false negative error rates for false positive ones as shown in Fig. 4.12 for $N = 100$ training points and a ratio of loss factors, $\frac{\lambda_{12}}{\lambda_{21}}$, of 100 that biases the decision toward negative classifications. The loss factor can be set interactively or with a predetermined formula such as Eq. 4.1

$$\frac{\lambda_{12}}{\lambda_{21}} = \frac{1000}{\sqrt{N}} \tag{4.1}$$

This formula was arrived at by trial and error with the intent of bringing the false positive misclassifications to zero at every stage of sampling. Figure 4.13 confirms this result, where the false positive misclassification is less than 3%, and the spike in false positive error rates seen in the early stage of sampling has been completely eliminated.


Full Connected BN Classifier without Losses

Fully Connected BN Classifier with Losses

Figure 4.12. The Loss Factor's Effect on the Systems' Decision Boundary

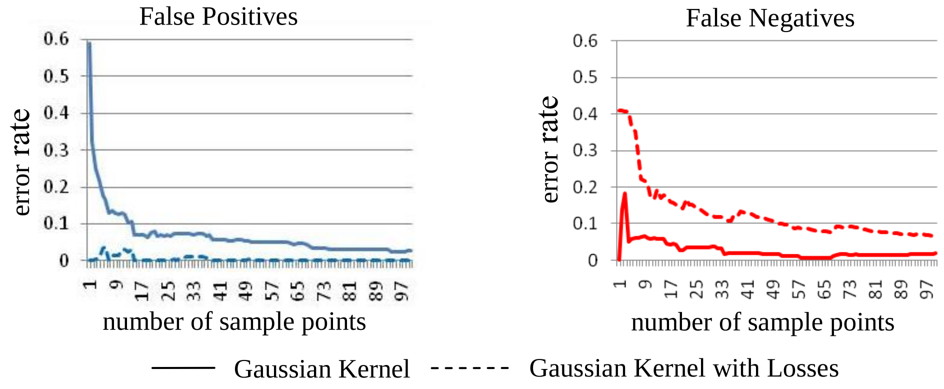


Figure 4.13. KBN Classifier Error Rates for the Systems’ Design Space

These results demonstrate that loss factors can be used to set the false positive error rates to zero. This capability is important for the case when it is simply too costly to have a misclassification. As more knowledge is gained, the loss factors can be adjusted to move the decision boundary closer to the correct boundary. This is a very important property for classifiers that are used in mechanical engineering design where the infeasible region often means mechanical failure. Loss factors can also be used to set the false negative error rates to zero. This capability is important for the case when the classifier is being used to identify promising regions of the design space for further exploration and potentially good designs should not be excluded.

4.4 DISCUSSION

The UAV design process presented in this chapter demonstrates the fundamentals of KBN classifiers and their ability to represent complicated, acceptable design regions for the purposes of coordinating collaborating design groups according to set-based design principles. What wasn’t considered in this design process is the final selection of a design, before proceeding with the design process. Instead, what was presented was the identification of a smaller region of the design space that should contain the best performing designs that are acceptable to all design groups. The final design can be chosen from this reduced region with a high probability of meeting all design requirements by sampling the class conditional probability distribution of the acceptable class. This is a fundamental goal of set-based design: to identify the mutually acceptable design space region within which to restrict future designs such that downstream iteration due to infeasibility is avoided. The belief is that the additional time spent generating the more thorough understanding of the design space relative to point-based methods will ultimately save time due to avoiding expensive later term iteration.

Other design processes than the one presented here are of course possible. In general there is no limitation on the order in which the design groups perform their classifications. For example, if the subsystem group had a very restrictive set of local constraints, they could classify the acceptability of their design space first, producing a classifier over the drag and weight variables that could be used at the system level as a check of feasibility in addition to having a high enough range. Because not all of the combinations of drag and weight that define the systems' search domain will be achievable by the subsystems group, future work might identify methods for choosing unachievable design points in order to construct a meaningful classifer that originates from the subsystems group. As it turned out in this example, the entire subsystem search domain was acceptable to the subsystems teams. In general it will be more efficient to identify the most restrictive requirement first and to use it to classify the acceptable regions. This will allow other teams to restrict their search over a smaller region of their design space. This performance gain will be investigated further in Chapter 7 where the possibility of adaptively sampling the design space is developed. What would happen if there is no mutually acceptable region of the design space? For example, what if the systems group needed a range of 1000 km? In this case, the systems group would have to adjust their constraints such that a mutually feasible region is found. Another possibility exists that is left for future work: the subsystem team could classify their region into the best and worst performing categories as defined by their proximity to the system level decision boundary or according to a measure of performance.

The design process of the preceding example could have been executed in a more concurrent fashion by allowing each group's classifiers to be shared at every stage of sampling. It is evident that allowing another team to use a higher error rate classifier will result in that team also having a less accurate local classification. It is not an issue in this case because the results of the local classifier are not used to adaptively select the next design points. Instead, the design points are independently predetermined as a space filling sequence. By the time both teams have sampled all 100 design points, the error rates of a more concurrent design process will be identical to the error rates of the presented serial process. The concurrent sharing of less accurate classifiers would become an issue for adaptive sampling when the results of the local classifier are used to determine the next sampling points. In this case, the adaptive sampling would not be as efficient because inaccurate information would be used to make the decision concerning where to sample next. This issue will be discussed further in Chapter 7 when adaptive sampling is developed.

In this discussion, the two subsystem teams, structures and aerodynamics, were merged into a single team for this baseline approach. This simplification was achieved by coordinating the values of their shared design parameters. Chapter 8 demonstrates a design process that decouples the training points between structures and aerodynamics, allowing each subsystem design team to sample their shared design space independently. This consideration of using classifiers for horizontal coupling provides an even greater flexibility in the scheduling of design activity at the potential cost of less accurate classifiers.

In addition to these process considerations, this research considers other methods for designers to construct their classifiers. Chapter 5 presents a novel method for automating the choice of kernel standard deviation. Although this will come at the cost of increased computational expense, it will eliminate additional time lost in experimentally finding a good rule for setting the standard deviations. Chapter 6 presents two simple ways to use knowledge of monotonic relationships for building classifiers with significant performance and accuracy gains over the baseline method presented in this chapter.

Chapter 5. Adaptively Setting the Standard Deviation

Chapter 3 introduced the KBN classifier and identified the standard deviation of the kernels as the most important setting affecting the classifier's performance. In the previous chapter, a formula was used to set the standard deviations. This approach provided good performance at the expense of using experimentation to tune the formula until it performed as expected. In this chapter, a novel method is developed that adaptively sets the standard deviations of the KBN classifiers automatically, avoiding the need for initial experimentation. The proposed method finds the largest standard deviations that still provide an acceptable level of posterior class probability for the training set. The new approach directly captures the expectations of a designer that 1) the classifier will correctly classify all of the training data, and 2) the smoothness of the decision boundary can be controlled in order for the classifier to interpolate well to unexplored designs provided that the first expectation is not violated. The proposed method is essentially an inexpensive automation of the process that was used to empirically arrive at the formula that set the standard deviations in the previous chapter. Furthermore, the method's results can suggest better formulas. Before providing the details of the proposed method in Section 5.2, the existing approaches to determining the standard deviations are discussed next. In Section 5.3, the proposed method is demonstrated on the UAV design problem introduced in Chapter 4. The final section is reserved for a discussion.

5.1 BACKGROUND

A designer using a classifier to find the boundary between feasible and infeasible regions of the design space will expect the training points to be correctly classified. However, depending on the location of the training points and the standard deviations of the normal kernel this may not always be the case. Figure 5.1 shows the result of using a fully connected classifier for the systems design problem with the scale factor, α_c, of Eq. 3.16 set to 1. From this example, one can see how overly smooth density estimates can lead to misclassifying a training point. This observation led to the experimentation in the previous chapter that resulted in using the smaller scale factors of Table 4.2. The goal of this chapter is to provide a means for automating the setting of the standard deviations such that all of the training points are correctly classified, removing the burden of experimentally determining good standard deviations from the designer.

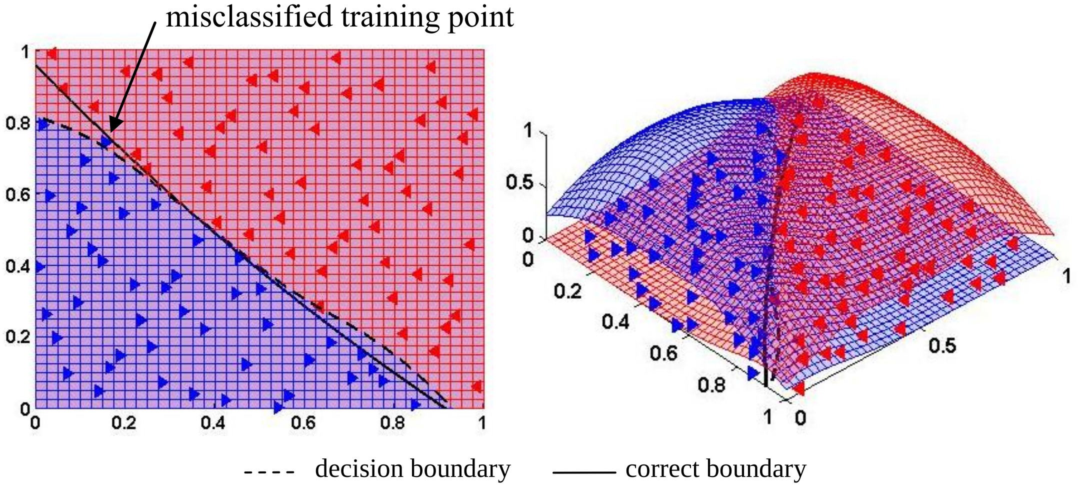


Figure 5.1. The Effect of the Standard Deviation on Classification of the Training Points

The challenge of setting the standard deviations in kernel density estimation is well known in the literature and many methods have been proposed. The majority of the methods are motivated by true density estimation problems that seek to estimate qualities of the targeted correct probability distribution that generated the training data (Silverman, 1986; Scott, 1992; Siminoff, 1996; Bowman and Azzalini, 1997; Sheather, 2004). The formula that was used to set the standard deviations in the previous chapter is tied to arguments that follow this motivation. However, the acceptable regions of the design space classified in this research are most likely not produced by probability distributions but by inequality constraints over the outputs of simulations. Hence the emphasis should be on producing low classification errors and not on density estimation. Accordingly, the following discussion reviews the methods for density estimation only to the extent that they inform the use of formulas such as Eq. 3.16.

The use of a formula to set the standard deviations is attractive because it is computationally cheap. But how can one arrive at an effective formula with a minimum of experimentation? The formula should have some properties in order to preserve the strong point-wise consistency of kernel density estimates: 1) the standard deviation, σ, must reduce to zero as the number of training points, N, increases to infinity, and 2) the product, $N\sigma$, must go to infinity as the number of training points, N, increases to infinity (John and Langley, 1995; Perez et al., 2009). Beyond these basic requirements, Eq. 5.1, called the Normal reference rule, can be derived from minimizing the asymptotic mean integrated square error for approximating a multivariate Gaussian distribution using Gaussian kernels (Silverman, 1986). Notice that N_c in Eq. 5.1 refers to the number of points in each class whose separate class conditional PD's are independently estimated, leading to two standard deviations, $\sigma_{i,c}$, for the acceptable and unacceptable classes, c, for each i of D dimensions. The $\hat{\sigma}$ is the standard deviation of the training points.

$$\sigma_{i,c} = \left(\frac{4}{D+2}\right)^{1/(D+4)} \frac{\hat{\sigma}_i}{{N_c}^{1/(D+4)}} \tag{5.1}$$

The first term on the right side of Eq. 5.1 is always relatively close to one, and hence Scott's rule, Eq. 5.2, is often used (Scott, 1992).

$$\sigma_{i,c} = \frac{\hat{\sigma}_i}{{N_c}^{1/(D+4)}} \tag{5.2}$$

It is also common to scale the data by each dimension's standard deviation, $\hat{\sigma}_i$, such that Scott's rule reduces to Eq. 5.3 (Perez, et al., 2009; Siminoff, 1996).

$$\sigma_c = \frac{1}{{N_c}^{1/(D+4)}} \tag{5.3}$$

Because the data used to explore the design spaces in Chapter 4 was chosen to uniformly fill a rectangular region of the design space of known extents, each dimension was scaled by the width of the interval defining the search domain instead of the standard deviation. Equation 5.4 is Scott's rule scaled by the relationship between the standard deviation of a uniform distribution and the width of the domain over which the uniform distribution is nonzero. Note that this relationship between the uniform distribution and the data's standard deviation is approximate because the data from each class fills only a portion of the total search domain. Later, this will be corrected for by using a scale factor.

$$\sigma_c = \frac{1}{\sqrt{12}{N_c}^{1/(D+4)}} \tag{5.4}$$

For the initial experimentation performed for the UAV design problem, Eq. 5.4 did not reduce the standard deviation fast enough in order to correctly classify all of the training data as the number of training points increased. Instead, a scaled version of the heuristic used by (John and Langley, 1995) and repeated here as Eq. 5.5 was found to perform better on the training set than Scott's rule. Note that (John and Langley, 1995) used $\alpha_c = 1$.

$$\sigma_c = \frac{\alpha_c}{\sqrt{N_c}} \tag{5.5}$$

Eq. 5.5 requires the two scale factors, α_c, to be set through initial experimentation. Notice that, unlike in Eq. 5.4, the rate of decay of the standard deviation with respect to the number of training points in Eq. 5.5 is independent of dimension. However, John and Langley (1995) use the same decay rate for classification problems with 4-19 continuous dimensions. The problems used by John and Langley (1995) to verify their classifier are different than the problems considered in this research. Nevertheless, their decay rate worked well on the UAV wing design problem as long as some experimentation was performed to set the scale factor. In subsequent sections of this chapter, the effectiveness of Eq. 5.5 is compared to the proposed adaptive method for setting the standard deviations, and a new rule is suggested that is better suited to the uniformly distributed training points used in this research.

Adaptive methods for setting the standard deviation based upon minimization of an estimate of the classification error are common in the literature for KDE-based classifiers (Dudda et al., 2001; Jain and Ramaswami, 1988; Specht and Romsdahl, 1994; Babich and Camps, 1996; Georgiou et al., 2006). Of the proposed methods, a search that minimizes the *K*-fold cross-validation estimate of the classification error is the most common approach. *K*-fold cross-validation is a technique that efficiently uses the data to estimate how well the classifier's performance generalizes to points that it has not been trained on. How it works is that the model is repeatedly trained *K* times using *N-N/K* of the data and tested using the remaining *N/K* data points. The final estimate of the model's performance is the average of the *K* train and test cycles (Dudda et al., 2001). This approach is summarized in Eq. 5.6. When *K* is equal to *N* the method is called leave-one-out cross-validation, LOOCV.

$$\sigma = argmin\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{l=1}^{N/K} err_{k,l}\right) \tag{5.6}$$

Cross-validation of the error is necessary because KDE smoothing parameters can be made arbitrarily small in order to achieve zero classification errors on the training set. This is undesirable because if the kernels are too sharp, then the classifier will not generalize well to interpolating points that are not close enough to an existing training point of the same class (Dudda et al. 2001). Cross-validation enforces smoother kernels that generalize as well as possible to the withheld training points. However, when applied to the UAV wing design problem, the resulting standard deviations that minimized the LOOCV errors did not necessarily result in a final classifier with zero misclassifications of all of the training data. Misclassification of any of the training points is not consistent with the prior knowledge of a designer who is certain of their simulation results. The proposed method that is developed next approaches this problem more from the point-of-view of the designer by guaranteeing that the classifier correctly classifies all of the training data to a specified level of confidence. The proposed method also seeks to find standard deviations that will generalize well to new data points, however not at the expense of misclassifying any of the training data.

5.2 THE PROPOSED METHOD

As mentioned earlier, the classifications in this research are usually a sharp definition of the acceptable region's boundary that comes from inequality constraints defined over the outputs of simulations. Therefore a boundary will usually exist that can exactly separate the data into the two classes: acceptable and unacceptable. Even with a probabilistic characterization of failure, a sharp decision boundary is often chosen that represents some acceptably low probability of failure. KDE classifiers are well suited to these spaces because there are standard deviations small enough to correctly classify all points with as high of a posterior class probability as desired. The posterior probabilities for the two classes must sum to 1. The decision surface defined by Eq. 5.7 in terms of the posterior probabilities must move between 1 for a high posterior probability of an acceptable design point to -1 for a high posterior probability of an unacceptable design point. The decision boundary is where the decision surface of Eq. 5.7 is zero. This surface is shown on the right side of Fig. 5.2 for the fully connected KBN classifier of the systems level with 100 training points from Chapter 4.

$$\frac{p(\hat{\boldsymbol{x}}^{\boldsymbol{k}}|c_1)P(c_1)-p(\hat{\boldsymbol{x}}^{\boldsymbol{k}}|c_2)P(c_2)}{p(\hat{\boldsymbol{x}}^{\boldsymbol{k}}|c_1)P(c_1)+p(\hat{\boldsymbol{x}}^{\boldsymbol{k}}|c_2)P(c_2)} \tag{5.7}$$

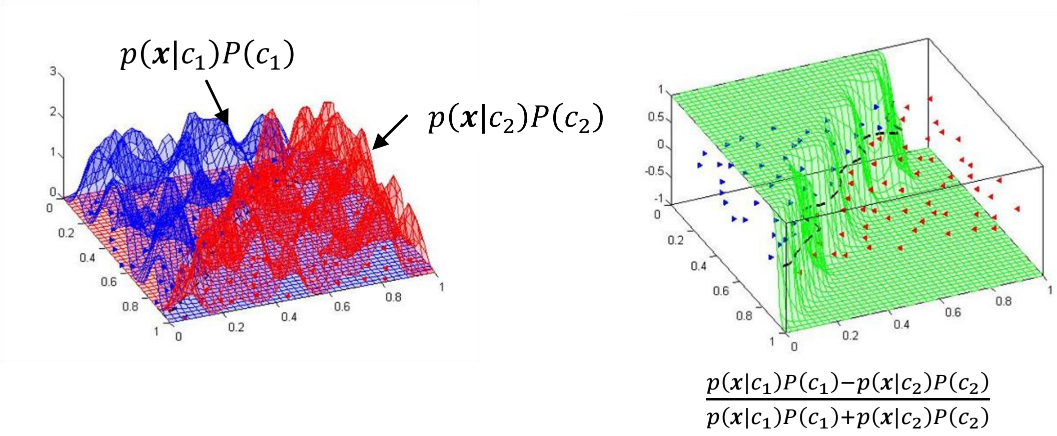


Figure 5.2. The Posterior Probability Decision Surface

These observations suggest a new approach: start from very small standard deviations and increase them until a training point is no longer classified with the desired posterior probability. If the desired posterior probability is 1, then the resulting standard deviations will be small. However, the lower the posterior probability threshold, the smoother the classifier will become so long as all of the training points are still correctly classified to the desired degree. If the posterior probability threshold is set to 0, then the largest standard deviations will be found that still correctly classify all of the data. Using this approach allows the user to indirectly set an upper bound on the standard deviations with a guarantee that all of the training data will always be correctly classified. The reason for not always choosing a threshold posterior class probability of 0 is that smoother class conditional probabilities do not always generalize best to untested design points. How well a classifier generalizes is dependent upon both the problem as well as the sampling sequence. However, the hope is that the performance of the proposed adaptive method will be insensitive to this new tuning parameter, and that the classifier will reliably behave in a manner more consistent with the knowledge of the designer. This is the essence of the approach that will now be developed in detail.

The proposed method is straight-forward. The user supplies a positive "confidence" level, *c*, which is the posterior probability of the class given the design point above which all training points must be correctly classified. Given the standard deviations and *c*, the error of misclassification is calculated according to Eq. 5.8-5.10. Notice that the computational complexity of calculating the error of Eq. 5.8 is equal to N times the complexity of evaluating the class as presented in Chapter 3: $O(N^2L)$, $1 \leq L \leq D$.

$$err = \sum_{j=1}^{N} err_j{}^2 \tag{5.8}$$

If the data point, $\hat{\boldsymbol{x}}^{\boldsymbol{j}}$, is from class c_1:

$$err_j = max\left(0, c - \frac{p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_1)P(c_1) - p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_2)P(c_2)}{p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_1)P(c_1) + p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_2)P(c_2)}\right) \tag{5.9}$$

If the data point, $\hat{\boldsymbol{x}}^{\boldsymbol{j}}$, is from class c_2:

$$err_j = max\left(0, c + \frac{p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_1)P(c_1) - p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_2)P(c_2)}{p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_1)P(c_1) + p(\hat{\boldsymbol{x}}^{\boldsymbol{j}}|c_2)P(c_2)}\right) \tag{5.10}$$

The search for the largest standard deviations that minimize the error is not well defined if the two classes are permitted to have two different standard deviations. This is because the posterior probabilities can be made arbitrarily high by making one class's standard deviation as small as necessary for any given fixed value of the other class's standard deviation. However, by enforcing the standard deviations of the two classes to be equal, the problem is well defined and can be solved through a line search method. This strategy was implemented using a customized golden section line search algorithm that begins at a very low standard deviation with zero error and increases the standard deviations until the error is no longer zero. The two standard deviations that bracket the zero error threshold are then refined using golden section search until the desired accuracy is achieved. The Matlab® source code implementing this procedure is in Appendix D. The total computational complexity of the proposed method will be the number of search iterations, M, times the complexity of calculating the error of Eq. 5.8, and could be quite expensive: $O(MN^2L)$, $1 \leq L \leq D$. This cost perhaps helps to justify simplifying the problem to a line search.

The proposed method is used in the next section to automatically set the standard deviations for the UAV wing design problem introduced in the previous chapter. The results suggest that low error rates can be achieved for the full range of values of the confidence: $0 \leq c \leq 1$. This is not the case for the scale factor, α, of Eq. 5.5 for which there is no well defined range of good performance. The generalization of the classifier over the range of settings for c is discussed in more depth in Section 5.4 in light of the results from the solution to the UAV design problem presented next.

5.3 UAV SOLUTION 3: BASELINE PLUS ADAPTIVE STANDARD DEVIATIONS

In this section, the same demonstration UAV wing design problem from Chapter 4 is solved again with only one difference: the proposed method for adaptively setting the standard deviations from the previous section is used instead of the previous rule-based formula. Three confidence levels of c = 1, 0.5, and 0 were used to classify the systems' design space. The resulting standard deviations are presented in Fig. 5.3 where they are compared to the rule-based standard deviations of Eq. 5.5 used in the previous chapter as well as Scott's rule of Eq. 5.4. From Fig. 5.3 it is evident that increasing the acceptable

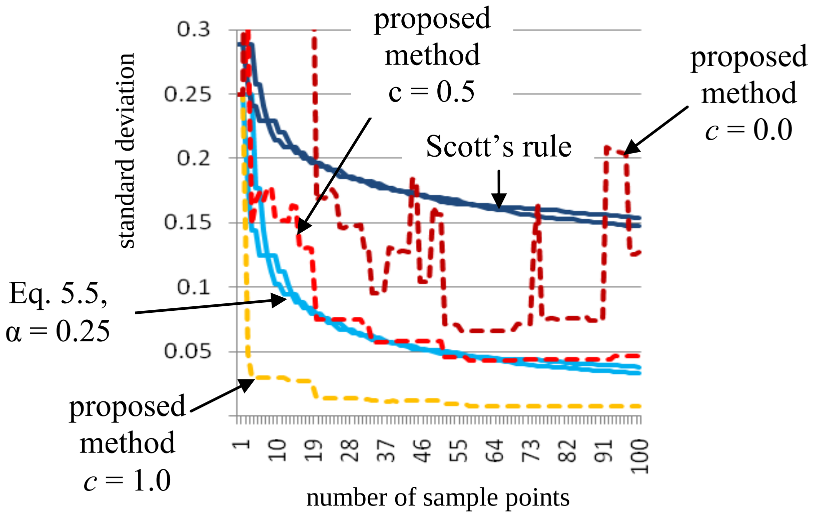


Figure 5.3. The Proposed Method and Rule-Based Standard Deviations

posterior class probability, *c*, does result in smaller standard deviations. However the functional form of the standard deviations with respect to the number of training points is not as smooth as the rule-based formulas, and the step changes appear to occur more frequently at the lower settings of *c*. This is perhaps the result of new training points having a greater probability of influencing the decision boundary as well as having a more significant influence on the decision boundary when they have a higher standard deviation kernel. For the sharper kernels that result from insisting that every training point is classified with a $c = 1$ posterior class probability, a new training point will most likely not be close enough to the decision boundary or to have a large enough of a nonlocal affect on the decision boundary to change the classification error and hence force a change to the standard deviation. This observation suggests that the adaptive setting of the standard deviations need not occur every time a new point is found. Perhaps the posterior probability of a new point can be used to determine if a line search for new standard deviations is necessary or not. This observation also suggests that the results can be sensitive to the sampling method, an effect that future work should seek to better understand.

The other encouraging result from Fig. 5.3 is that the empirical conclusion in Chapter 4 to not use Scott's rule and to instead use the faster decaying formula of Eq. 5.5 appears to be justified by the proposed adaptive search method. The justification comes from the fact that the $c = 0$ curve is often below Scott's rule which means that Scott's rule is often misclassifying some of the training points. But the $c = 0$ curve is above the Eq. 5.5 curve which means that the Eq. 5.5 curve led to no misclassifications of the known data throughout the experiment. For a designer observing the progress of the classifier, a misclassification of a training point whose correct class is known with certainty would not make sense and would lead them to look for new values for the standard deviations.

For the solution presented in Chapter 4 this led to the use of Eq. 5.5. The proposed method achieves this search automatically, and the adaptive method with c = 0.5 produces standard deviations that are quite similar to Eq. 5.5 with a scale factor, α, of 0.25.

When adaptive methods were being explored to set the standard deviations, LOOCV was tried, producing standard deviations that are even higher than Scott's rule, as shown in Fig. 5.4. Because LOOCV misclassified some of the training points, a new adaptive method was sought that explicitly did not misclassify any of the training points, leading to the proposed method. The standard deviations produced by LOOCV not only were too large, they were also very erratic, especially at low sample sizes. The sensitivity of LOOCV to the training points has been reported in the literature and is another reason to look for an alternative adaptive method such as the proposed method (Jain and Ramaswami, 1988).

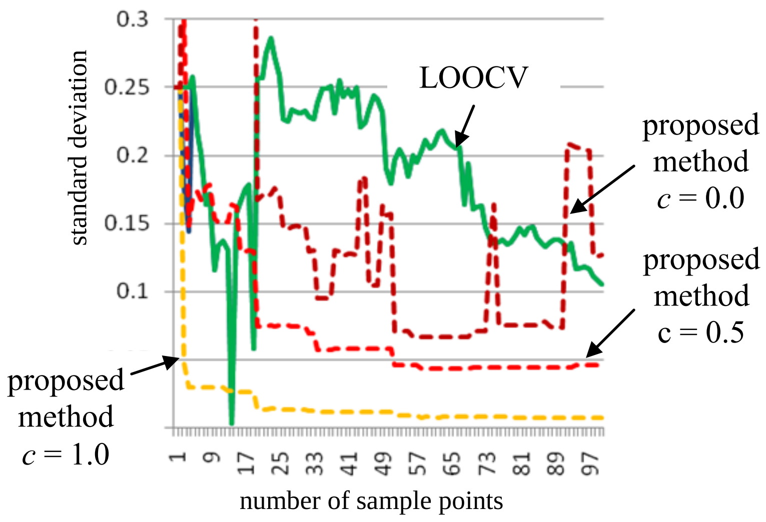


Figure 5.4. The Proposed Method and LOOCV Standard Deviations

A final point of interest is that there are two standard deviation curves for Scott's rule as well as for Eq. 5.5 in Fig. 5.3. This is because the formulas are based upon the number of points in each class which is not necessarily the same but depends upon the correct decision boundary and the sampling sequence. For the systems' design problem, the number of points in each class is very similar and the two curves for each rule are hence also very similar. As will be seen soon, this is not the case for uniform sampling over a domain within which the acceptable design space is a much different size than the unacceptable design space region, such as occurs in the subsystems' design problem.

Before presenting the same results for the subsystems' problem, the question of which of these methods for setting the standard deviations is best in terms of classification errors will be addressed for the systems' problem. Just the same as before, 100 training points from the Halton sequence were used to train the classifiers which were subsequently tested using 1000 different design points from the Hammersley sequence whose correct classification was determined by the systems' simulation. Fig. 5.5 shows the total classification error rates: the false negative plus the false positive error rates. From these results there is no single conclusively better strategy for setting the standard deviations with respect to the total error rates, and the resulting performance of the different classifiers is not sensitive to the choice of strategy. Strategies that appear to perform better with fewer training points do slightly poorer with more training points. By about 15 training points, all strategies have converged to error rates below 0.15 and they all agree within a range of about 0.05. Thus the final accuracy of the classifiers that

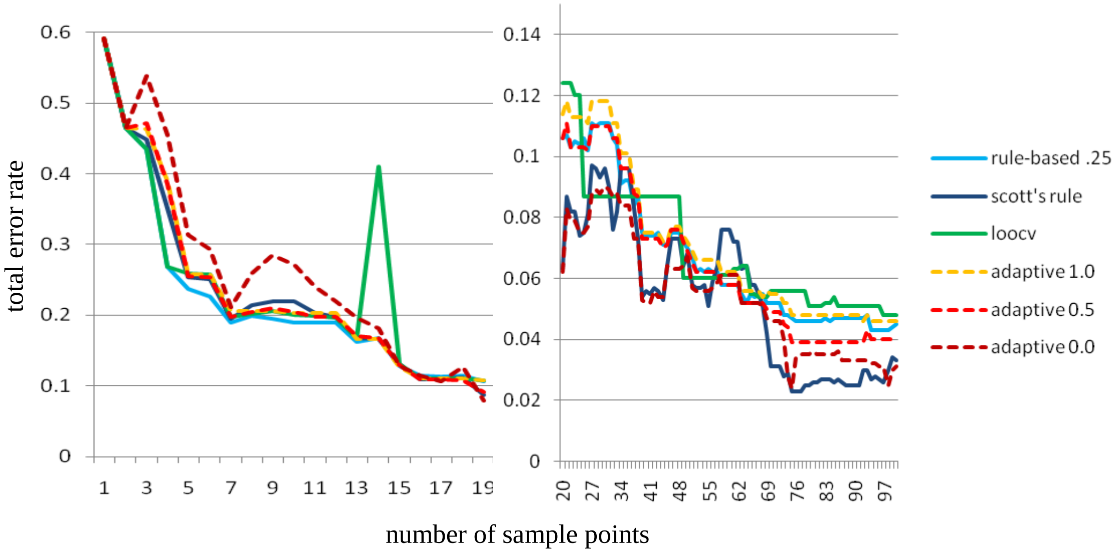


Figure 5.5. Systems Level Total Classification Error Rates

use the proposed adaptive search to set the standard deviations are relatively insensitive performance-wise over the whole range of settings for *c*. The LOOCV method led to large error for the set of 14 training points when the standard deviations became very low, emphasizing the sensitivity of this method to the training points. The rule-based methods provide equally accurate classifiers even though they may misclassify some of the training points and might appear to be inaccurate to the designer constructing the classifier. The last conclusion will not hold for applying Scott’s rule to the subsystems’ problem presented next.

These experiments were repeated for the subsystem design problem and the results are reported in Fig. 5.6-5.8. The total error rate of the subsystem classifiers was found using the correct classification as determined directly by the systems level simulation and not indirectly using the systems’ classifier as was done in Chapter 4. Presenting the total error rates with respect to the systems’ simulation allows for a clearer comparison of how well the classifiers perform on the subsystem's design problem without the additional variability of an inaccurate systems' classifier.

For the subsystems' problem, the proposed method successfully finds standard deviations automatically that result in well performing classifiers. Scott's rule does not. In contrast to the systems' problem, the subsystems acceptable and unacceptable regions are of very unequal size, and because the samples are drawn roughly uniformly across the entire design space, there are very unequal sample sizes for each class. This leads to two very different standard deviation curves for Scott's rule in Fig. 5.6. By about 15 samples, this imbalance leads to standard deviations that are too large for the acceptable class, and consequently the acceptable class conditional PD is too low relative to the sharper class conditional PD of the unacceptable region to ever produce a classification boundary at all—the entire design space is classified as unacceptable. This explains the error rates in Fig. 5.8 for Scott's rule leveling out at 0.09 all of which are false negative errors that occur in the acceptable region that is about 9% of the search domain.

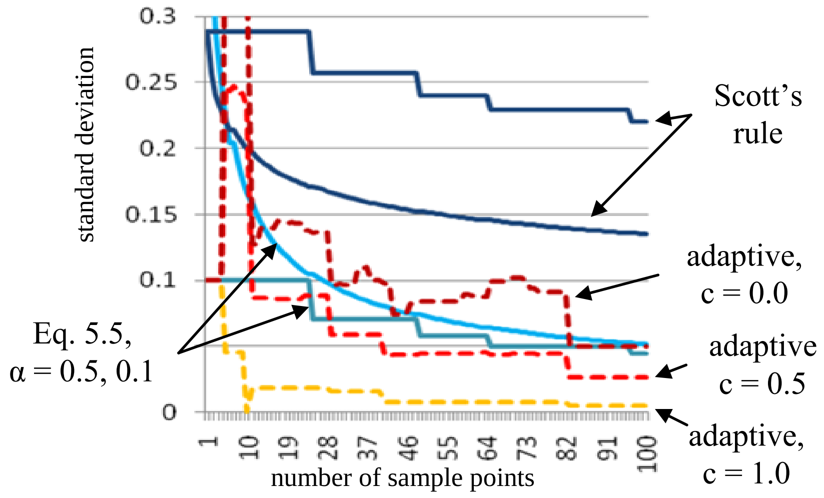


Figure 5.6. The Proposed Method and Rule-Based Standard Deviations

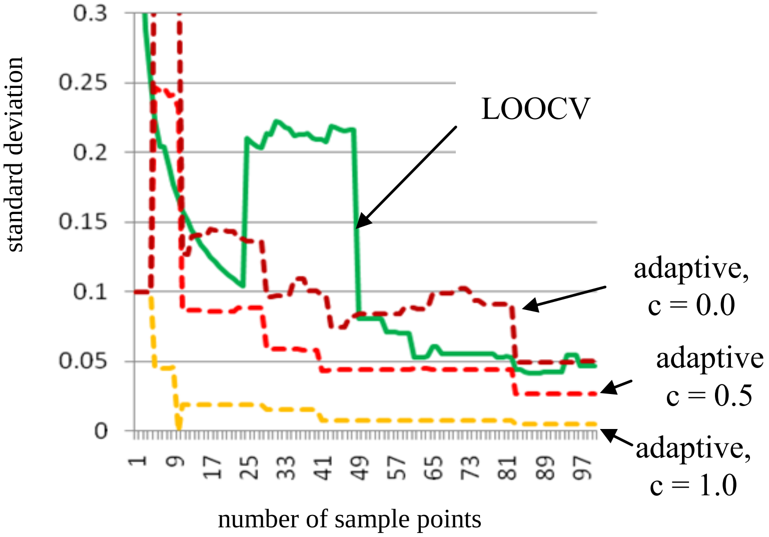


Figure 5.7. The Proposed Method and LOOCV Standard Deviations

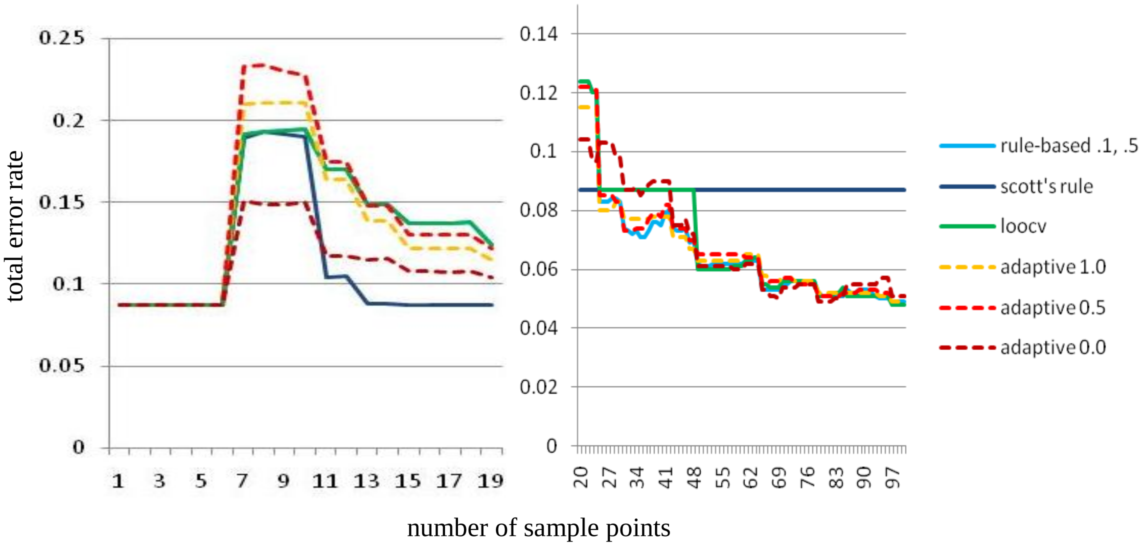


Figure 5.8. Subsystems Level Total Classification Error Rates.

When searching for an adequate formula to set the standard deviations in Chapter 4, this inadequacy of Scott's rule was observed. When Eq. 5.5 was tested as an alternative, experimentation led to choosing two different scale factors of α = 0.1 and 0.5 for the acceptable and unacceptable classes respectively. The difference in scale factors compensated for the difference in sample sizes, producing roughly similar standard deviations as shown in Fig. 5.6. Using the proposed adaptive method for setting the standard deviations automatically created standard deviations that are the same for each class and are similar in magnitude to the two Eq. 5.5 curves with the two different scale factors. After about 20 samples, all of the methods except Scott's rule produce similar error rates. As shown in Fig. 5.7, LOOCV produces standard deviations that are larger than the proposed method with $c = 0$ for a portion of the training point sets. Hence, LOOCV does not correctly classify all of the training points for the subsystem design problem. The failure of LOOCV to be consistent with the training points for the systems and subsystems spaces led to the development of the proposed method.

There are four conclusions for these design spaces: 1) the proposed method can automatically find a good sequence of standard deviations that leads to well performing classifiers over the full range of its tuning parameter, 2) formulas exist that create classifiers that perform just as well as the adaptive search for standard deviations and will have shorter training times, 3) the proposed method can replace the experimentation required to adjust the formulas to perform well or to perform as expected by the user but with a penalty of increased training time, and 4) the proposed adaptive search method can suggest the use of a new formula in order to recover the benefits of a rule's fast training time. These conclusions will be discussed in more detail in the next section, but first the new formula suggested by these results is presented.

The new formula, Eq. 5.11, uses a single standard deviation for both classes that decreases with the total number of design points; an approach that is more consistent with the use of uniform sampling to generate the training points. The new formula also suggests starting with a scaling factor of 0.4 which should perform well for both the systems and subsystems problems.

$$\sigma = \frac{0.4}{\sqrt{N}} \tag{5.11}$$

Figure 5.9 compares the standard deviations generated from Eq. 5.11 to the results of the adaptive standard deviations. The similarity of standard deviations between the methods suggests that the error rates using the new rule will perform well. This is confirmed by the error rates reported in Fig. 5.10 for the new formula which are consistent with the error rates from the proposed adaptive method. The new rule-based classifier will be used in subsequent chapters as the baseline method against which further enhancements will be compared.

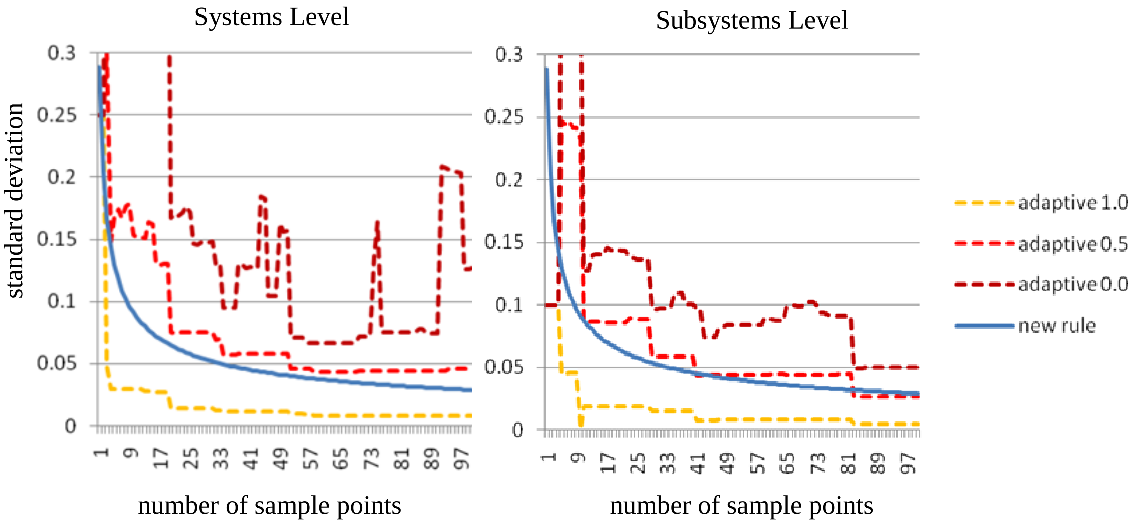


Figure 5.9. New Rule Standard Deviations

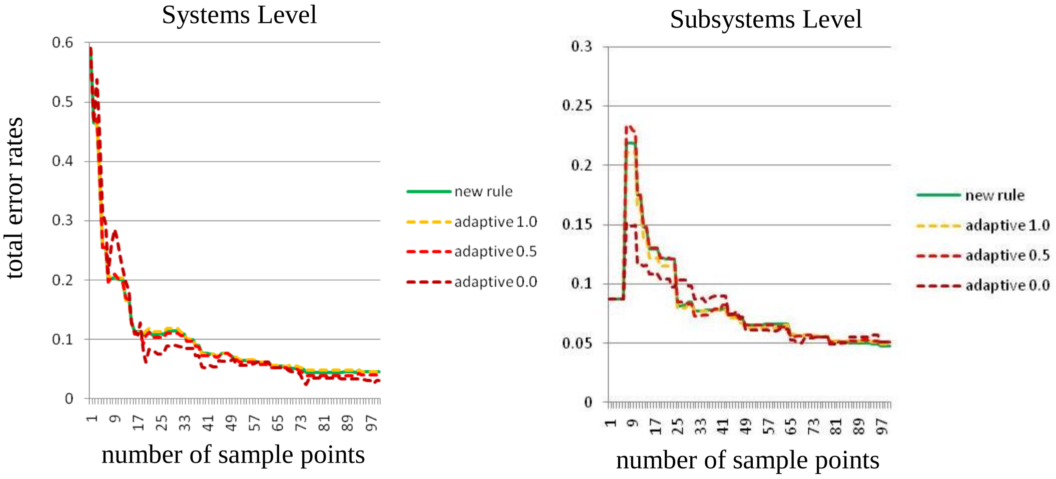


Figure 5.10. New Rule Total Error Rates

5.4 DISCUSSION

In this chapter, the choice of setting the standard deviations has been explored in detail. The literature suggests that the standard deviations are the most important setting for kernel density estimation. The classification of the system and subsystem design spaces confirms that it is important to the classifier's performance in two ways. First, the classification error rates can be high and not converge for some design problems such as the subsystems' design problem where the two classified regions had very different numbers of training points leading to complete misclassification of the acceptable region using Scott's rule. Second, although the error rates of some methods might not be much worse than other methods, these methods might misclassify their own training points. The latter concern is a less severe example of the first concern, but it highlights the importance of the prior expectations of the designer using the classifier. Typically a designer will not be aware of a classifier's error rates as determined by a large set of test points such as the 1000 Hammersley sequence points used to determine the total error rates reported above. A designer will only be aware of a contradiction to their available knowledge: a misclassification of one of the training points. The proposed adaptive method for setting the standard deviation avoided any of this latter type of error for all settings of its tuning parameter, *c*. Furthermore, increasing *c* resulted as desired in decreased standard deviations.

The ability to still control the sharpness or smoothness of a classifier acknowledges that different design problems as well as different training point sequences will generalize better according to two different assumptions. The first assumption is that the correct decision boundary is smooth. The second assumption is that the correct decision boundary should have the maximum margin between the closest training points on either side of the decision boundary. The second assumption means that, given the current training points, the correct decision boundary is more likely to fall half way between the points that lie closest to the decision boundary, i.e. to not favor an unnecessarily larger or smaller acceptable design space within the region that still correctly divides the training points into their two classes. These assumptions might be justified by a designer's a priori expectations of the design problem, or they might not. Therefore, it is reasonable to provide a new tuning parameter that can be set according to these expectations. The threshold acceptable classification probability, *c*, will produce smooth boundaries for lower settings and more marginal boundaries at higher settings. Figure 5.11 illustrates these trends for the systems' and subsystems' design spaces after 30 training points have been sampled. Importantly, the expectation of the designer to not misclassify any of the training points is not compromised for any setting of *c*. This means that for $c = 0$, there is likely to be a training point directly on the decision surface. This discussion suggests a possible direction for future work: to train three classifiers in parallel with $c = 0$, 0.5, and 1.0 and to use the classification with the majority vote or some other method of combining classifiers (Kuncheva, 2004). On the other hand, setting $c = 0.5$ appears to be a good default compromise between having a smooth as well as a large margin boundary. Other classifiers should also be considered that are more flexible and able to independently adjust classification boundary margin and smoothness.

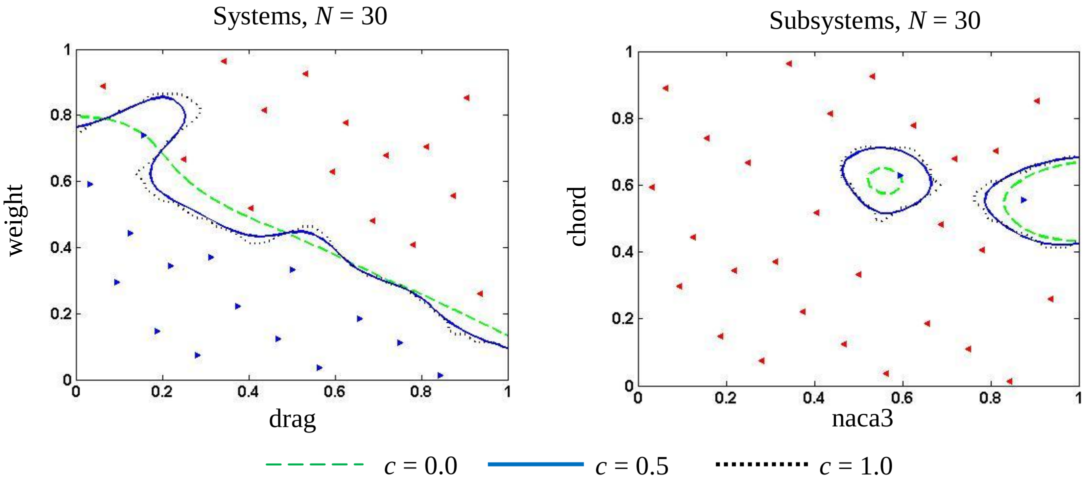


Figure 5.11. Decision Boundaries for $c = 0, 0.5, 1.0$

One can also conclude from this chapter that a rule can be created that leads to classification error rates that are equivalent to the proposed adaptive method and that benefits from a shorter training time. Equation 5.11 appears to be a good place to start for finding such a rule. If the new rule of Eq. 5.11 does not perform acceptably, one has recourse in the proposed adaptive method to produce a classifier that will perform consistently with the designer's knowledge at training time: it will not misclassify design points of known acceptability. Furthermore, the sequence of standard deviations created by the proposed adaptive method might suggest a new rule that will perform well for the new type of design space encountered. Future work along the same investigative vein will clearly require more experiments on different design spaces. In particular, problems of higher dimensionality will most likely require different decay rates, and Eq. 5.11

should be updated to reflect this dependency. Future work should also seek to understand the extent to which a rule's performance is sensitive to the sampling method. Some evidence of this affect is provided in Chapter 7 when adaptive sampling methods are developed.

As an aside, an interesting result that came from using the systems' simulation and not the systems' classifier to determine the acceptability of the subsystem design points is that the subsystems' acceptable region is classified as two disconnected islands of the design space as seen on the right of Fig. 5.11. While the correct acceptable region is really four disconnected islands as shown in Fig. 4.9, the sampling resolution was only able to find a single acceptable design point from the three islands other than the largest island. But for this isolated acceptable design the classifier was able to create a disconnected acceptable design region. This is a validation of a minor part of this research's hypothesis that the proposed method can classify arbitrary and *potentially disconnected* regions of the design space.

The disconnected islands of feasibility come from a nonobvious combination of 1) decreases in weight relative to increases in drag such that in the balance the range rises above the target level as thickness increases and 2) discontinuous jumps of higher drag for a few critical thicknesses that drop the range below the target level. Combining the results of multiple nonlinear simulations appears to easily produce such complicated feasible regions, although more example problems are needed to understand how frequently these types of feasible regions appear and how critical it is to accurately capture them. In this case, the smaller islands of feasibility have borderline acceptable ranges. The designs with the highest range exist in the largest island of feasibility.

This chapter essentially took a classifier from the literature and made it perform in a manner consistent with the prior knowledge of a designer using simulations or analytical models of their design problem to try to understand the acceptable combinations of their design parameters. The only strictly enforced assumed knowledge of the designer was the correct classification of a set of design points—the training set—that has been evaluated using a simulation. The designer will most likely possess other knowledge, too. In particular, knowledge of monotonic relationships between parameters is a common occurrence in mechanical design. The next chapter explores two ways in which the KBN classifier can exploit this type of knowledge for performance gains.

Chapter 6. Incorporating Knowledge of Monotonic Relationships

The use of classifiers for coordinating mechanical engineering design activity can benefit from easily incorporating expert knowledge to make the classification decision more efficient. The only information required is a judgment on whether a design point is acceptable or not. In previous chapters, this judgment came from the satisfaction or violation of inequality constraints on the outputs of simulations. However, mechanical engineering design is full of decisions that can be made without simulation and instead by exploiting an understanding of the relationships between variables that are based upon a physical understanding of the problem. For example, performance can almost always be improved if physical losses are minimized. For the example UAV wing design problem, it is clear that range will always improve with lower drag. As another example, the range will also always improve if the wing weight can be minimized because a lighter wing increases the portion of the total weight allocated to carrying more fuel. What is not immediately clear is how to coordinate the subsystems teams to achieve the best low weight and low drag design because the two subsystem problems are coupled. Nevertheless, this knowledge can significantly improve the efficiency of classifying the systems' design space.

Many methods exist that seek to incorporate expert knowledge into the decision making process. An important distinction between many of these methods and the proposed method of this chapter is that here there is no attempt to reconcile differences in opinion from multiple expert sources. Up until this point, it has been assumed that every design point has a single correct label. The only new possibilities introduced in this chapter are that 1) there may be multiple instances of a design point in the training set as long as they all have the same class label and 2) these design point labels do not need to necessarily come from a simulation. Allowing these possibilities opens up the classifier to having higher error unless care is taken to maintain the integrity of the data. The data should contain only the highest fidelity information available and there should be no conflict in classification for any given training point. A possible alternate approach reserved for future work is to train multiple classifiers based on each different source of information and to combine their classifications appropriately (Kuncheva, 2004). However, provided that the new information is accurate, there are two simple ways that are provided in this chapter by which the classifier already developed can be potentially improved in terms of lower classification errors with fewer simulation runs: 1) seeding the data with training points of known class and 2) not running a simulation to determine a new training point's class because its correct class can be determined accurately using our prior knowledge.

The type of reasoning discussed in the opening paragraph that allows one to assume that a parameter will always increase or decrease as the result of increasing or decreasing another parameter is known as monotonicity. Monotonicity is a powerful way of extrapolating decisions to new designs, and it is the only type of knowledge explicitly demonstrated in this chapter. Other forms of knowledge of course exist and could be similarly exploited to improve the classifier using either of the same two mechanisms of seeding the data and classifying without simulation. Before the details of the method are revealed in Section 6.2 and the results presented in Sections 6.3-4, other methods that take advantage of monotonicity are presented next and discussed in relationship to the approach taken here.

6.1 BACKGROUND

Leveraging monotonic relationships to make optimization more efficient was pioneered by (Papalambros and Wilde, 2000). In their work, inequality constraints can be determined to be active or not by careful reasoning about the monotonic relationships between variables. An active inequality constraint can be changed to an equality constraint, reducing the dimensionality of the resulting problem. The reasoning can be quite complex as the number of variables with monotonic relationships increases and tools such as monotonic influence diagrams have been created to automate the conclusion of which constraints are active (Michelena and Agogino, 1992).

This is a powerful approach that is used frequently to simplify a design problem. For the purposes of this research, using monotonic relationships to reduce the dimensionality of the search space should be incorporated into every group's search strategy as long as the resulting simplification is local and does not involve coupled design parameters. In fact, for the example UAV wing design problem the monotonic increase in wing stress with decreased wing weight is used to efficiently set the structures group's only hidden design parameter: the wing skin thickness. Thus for every (*naca3*, *chord*) design point, the structures group assumes that the lowest weight design will be the design with a stress that is equal to the acceptable wing strength limit and adjusts the skin thickness accordingly. This optimization occurs in the background unbeknownst to the aerodynamics and the systems groups.

However, if the monotonic relationship involves a coupled parameter, the approach of this research is not to reduce the dimensionality of the design space, but rather to use this knowledge to more efficiently map the boundary of the feasible region. This is justified by the observation that one group's monotonic reasoning can lead to setting a parameter equal to an active constraint that is not feasible to their collaborators.

This is also illustrated in the UAV wing design problem, where the systems group would conclude that the wing weight and drag should both be zero, which is clearly not an attainable goal for the subsystems groups. The belief is that by providing an acceptable design region as opposed to an acceptable design point, many good designs will be identified as opposed to a single design that is the best only locally or currently. This is consistent with the principles of set-based collaborative design.

Another method that also exploits knowledge of monotonic relationships includes qualitative and quantitative sequential sampling, Q2S2 (Rai and Campbell, 2008). While Q2S2 is primarily a sequential sampling method and this research is concerned with mapping and sharing feasible regions, how Q2S2 uses qualitative information such as monotonicity is informative. Q2S2 defines a confidence value that is a function of the design parameters as well as a performance parameter where a confidence of 1 means that the design point has the specified performance value with absolute certainty, -1 means that the design point does not have the specified performance value with absolute certainty, and 0 means that there is no information available about the design point's value. Monotonicity is used to influence the shape of the confidence field such that low confidence (≤ 0) is propagated to regions of the design space above or below a given design point, depending on the direction of the monotonic relationship. In this manner, monotonicity is used to reduce the search space for future design points. The equivalent approach for this research would be to directly manipulate the class conditional probability distribution at training time. As will be seen in the next section, the approach taken in this research is indirectly through the data points.

Q2S2 also has the capability of incorporating information from more than one source in a manner that is very similar to the other method proposed in this chapter: seeding the database with points of known class. However, Q2S2 allows for ranking the information according to a confidence measure and merging different confidence fields over the same design space. Hence Q2S2 can handle contradictory classifications at the same data point. As discussed earlier, this capability does not yet exist in the proposed method although there is a large literature on methods for combining classifiers that can achieve this capability (Kuncheva, 2004). More will be said about the similarities and differences between Q2S2 and this research in the next chapter where adaptive sampling is developed.

To the author's knowledge, these are the only examples of exploiting monotonic relationships in engineering design. In the classification literature, monotonic relationships have also been used to improve classification error. However, the classification literature usually takes the class of the training data as a given, not something to be discovered. Hence, exploiting monotonic relationships involves improving the training set (Duivesteijn and Feelders, 2008). In contrast, the proposed method exploits monotonic relationships to seed the training set and to avoid potentially costly evaluations of the correct class via simulation for new training points. The details of the proposed methods are provided next.

6.2 THE PROPOSED METHODS

The first method for incorporating prior class knowledge is simple: the training data set is filled with points of known class. These initial points will create a classifier that can potentially avoid the highly variable decision surfaces that occur with low numbers of training points. As more points are sampled and the standard deviations are reduced, the effect of the initial seeded data will become more local, and the newly sampled points will increasingly dominate the definition of the decision boundary. The rate at which the seeded points become less influential can be indirectly controlled by including more than one instance of a training point in the training set. This tactic increases the weight of a given design point relative to the other points by as many times as there are replicated instances. In order to allow these samples of enhanced weight to have their full influence, the formula for calculating the standard deviations should be shifted relative to Eq. 5.11 by the number of replications, N_r, according to Eq. 6.1.

$$\sigma = \frac{0.4}{\sqrt{(N - N_r)}} \tag{6.1}$$

The second method for incorporating expert knowledge is purely an efficiency gain: new samples are compared against the training set to determine if they are dominated by other points of known class or not. If they are dominated, they can be classified without running the point through the required simulation. A point is dominated by another point if all design parameters are greater than or less than the dominating point's design parameters according to the monotonic relationship between the design parameters and the performance parameter. Implementing this performance enhancement requires a single pass through two new arrays that store the acceptable and unacceptable dominating points that bound the classification decision from both sides. Every new sample point is compared to the points in the dominance arrays to check if it is dominated or not. If a point is dominated by an unacceptable point in a manner consistent with the acceptability constraint then it is classified as unacceptable and added to the training set. For the UAV systems-level design problem, this type of dominance means that both the drag and the weight of the new design point are larger than another design point that is known to be unacceptable. If the point is not dominated by any of the unacceptable points, then it is compared to the dominant acceptable points. If a point is dominated by any of the dominant acceptable points in a manner consistent with the acceptability constraint then it is classified as acceptable and added to the training set. For the UAV systems-level design problem this type of dominance means that both the drag and the weight of the new design point are less than another design point that is known to be acceptable. The check for dominance requires evaluating inequality relationships over all M monotonic variables with $1 \leq M \leq D$ where D is the number of dimensions. Hence the time complexity of the check is $O(MN)$, $1 \leq M \leq D$. If the point is not dominated by either of the dominant acceptable or unacceptable points, its class is determined by simulation. If the simulation classifies the new point as acceptable/unacceptable then the new point might dominate some of the points in the array of dominant acceptable/unacceptable points. In order to keep the arrays of dominant points to a minimum length, every new point that is classified not by monotonic reasoning but by simulation is compared to the dominant points in the array of the same class to see if any of the previously dominant points can be removed from the array because they are now dominated by the new point. This procedure is implemented in a Matlab® in Appendix E for the example UAV wing design problem.

The first method for incorporating expert knowledge by seeding the training set with points of known class is demonstrated next using the systems' UAV wing design problem. Section 6.4 will solve the system's classification again but using only monotonic reasoning to determine the class. Both methods are independently and favorably compared in terms of error rates as a function of the number of simulation runs to the new rule-based baseline solution presented at the end of the previous chapter.

6.3 UAV SOLUTION 4: BASELINE PLUS SEEDED DESIGNS

In this section, the systems UAV wing design problem of finding combinations of weight and drag that produce ranges that are greater than or equal to 900 km is solved again using the same classification method presented at the end of the previous chapter with the new rule of Eq. 5.11 to set the standard deviations but also using an initial seeding of the training set. The seeding is the consequence of monotonic reasoning about the effect of drag and weight on the UAV's range already mentioned in the opening paragraph of this chapter: decreasing both the drag and the weight will increase the range of the UAV over the entire search domain. As a consequence of this observation, one can safely say that the point of zero wing drag and weight will exceed the range requirement of 900 km while the point of maximum drag and weight will not. Being confident in this observation, one can give it a high weight by adding to the training set 20 design points that are not evaluated by the simulation: 10 instances of the design point (drag = 0, weight = 0) classified as acceptable and 10 instances of the design point (drag = drag_{max}, weight = $\text{weight}_{\text{max}}$) classified as unacceptable. With the number of replicate design points, N_r, set to 20, more points can be added to the seeded training set according to the same Halton sequence used in previous chapters and with the same sequence of standard deviations as provided by Eq. 6.1. The class of the new training points from the Halton sequence is determined by running the systems' simulation. The decision boundaries and class conditional probability distributions from 0, 10, 20, and 100 simulation runs are shown in Fig. 6.1. The salient observation is how the influence of the seeded information becomes more local with every additional new training point until it no longer has a significant effect on the shape of the decision boundary.

The false positive and false negative error rates for the seeded classifier are reported in Fig. 6.2 where it is compared to the new baseline classifier introduced at the end of the previous chapter. The error rates are now reported as a function of the number of simulation runs instead of the previous metric of the number of training points because the initial 20 seeded training points came for free; the real cost is in running a simulation.

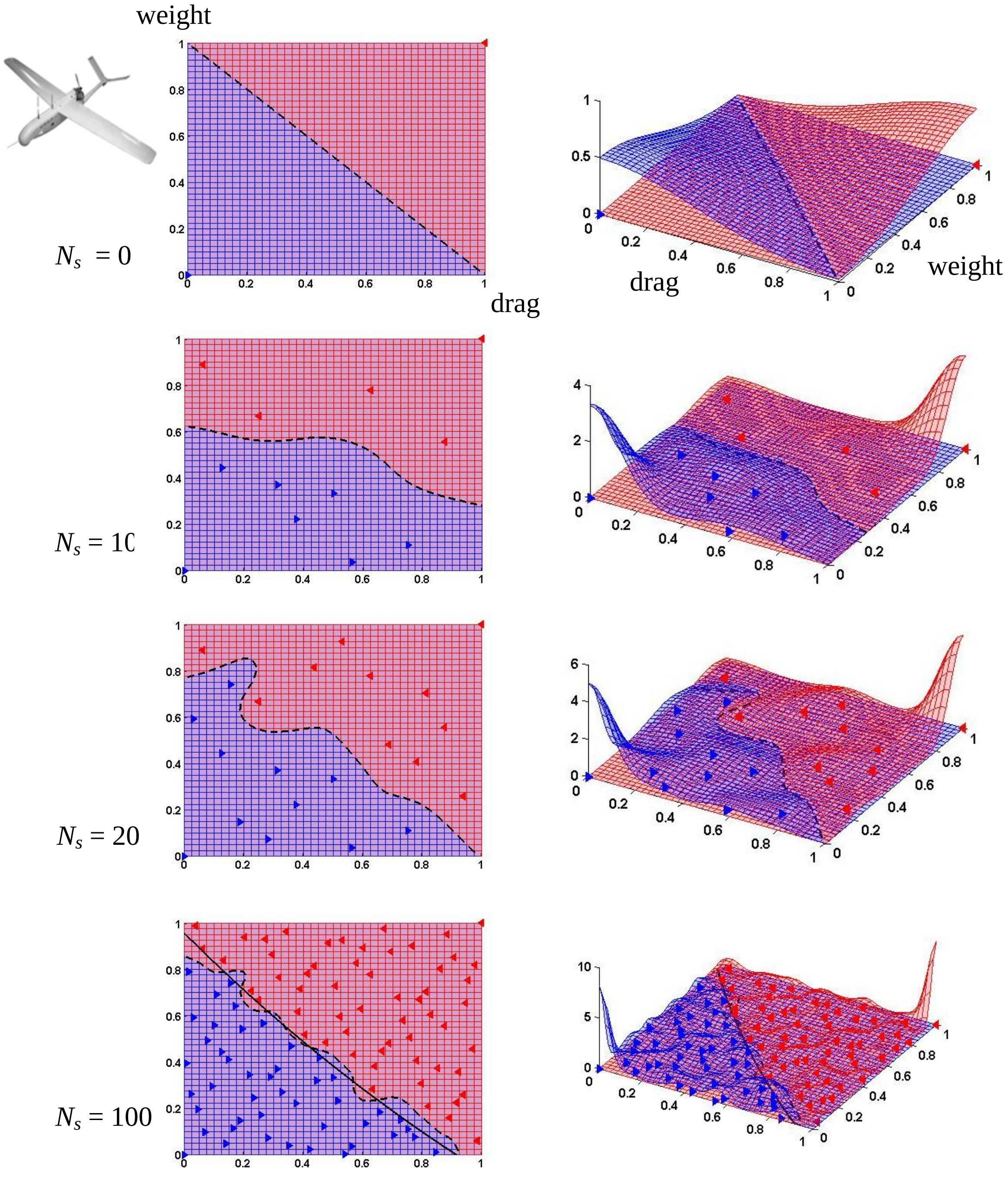


Figure 6.1. The Systems' Classifier for 0, 10, 20, and 100 Simulation Runs

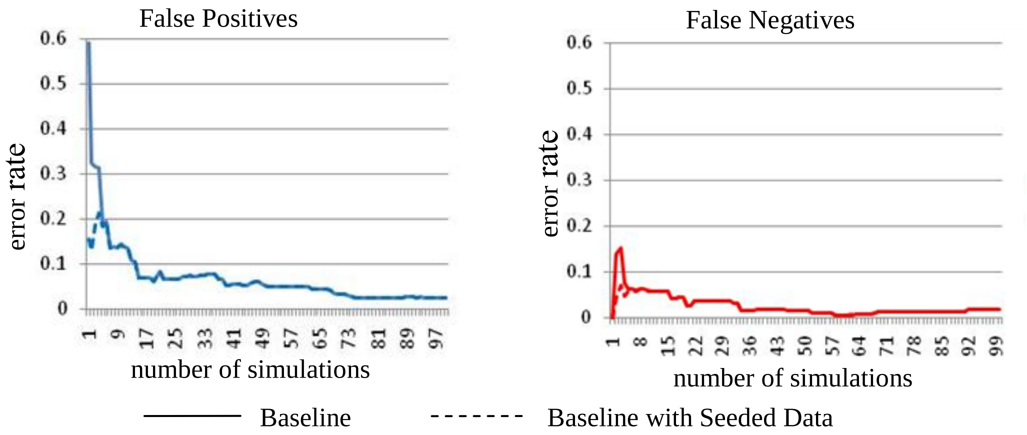


Figure 6.2. KBN Classifier Error Rates for the Systems' Design Space

The initial error rate is dramatically improved using the seeded classifier, although it must be recognized that in the presented example, the decision boundary was luckily near the half-way point. In general the decision boundary could lie anywhere between the two corners and hence the gain in classification error rates might not be as good as presented for the UAV systems-level design problem. This demonstrates the potential effectiveness of this first of the two proposed methods for incorporating knowledge of monotonic relationships; the second method is demonstrated next.

6.4 UAV Solution 5: Baseline Plus Automatic Classification

In this section, the second method for incorporating expert knowledge is used to improve the efficiency of the systems' UAV wing design problem: using monotonic reasoning to determine a new point's class without running a simulation. Every time a new design point is taken from the Halton sequence it is sequentially compared to all of the points in the dominant unacceptable design point array to see if the new point has a drag and weight that are both greater than the drag and weight of any of the dominant unacceptable points. If the new point is dominated in both drag and weight dimensions, then it is immediately classified as unacceptable and added to the training set. Otherwise, the new point is similarly compared to all dominant acceptable design points to see if the new point has a drag and weight that are both less than the drag and weight of any of the dominant acceptable points. If the new point is dominated in both drag and weight dimensions, then it is immediately classified as acceptable and added to the training set. Otherwise, the new design point's class cannot be determined by monotonic reasoning and its range has to be determined by simulation and classified as acceptable or unacceptable according to whether or not its range is greater than or equal to 900 km. Finally, for any point whose class is determined by simulation, it is compared to all of the points in the dominance array of the same class to determine if the new point dominates any of them. If so, the previously dominant point is removed from the dominance array. As mentioned previously, this keeps the dominance array to a minimum length and stores all of the points that are closest to the decision boundary for the monotonic dimensions. The decision boundaries and class conditional probability distributions from 10, 20, and 100 training points are shown in Fig. 6.3. The currently dominant points are the larger triangles. In Fig. 6.1 both the total number of training points, N, as well as the total number of simulation runs, N_s, is reported. The efficiency gain is demonstrated by requiring only 35 simulation runs for all 100 training points.

Figure 6.4 shows the error rates of the proposed method as a function of the number of simulation runs compared to the baseline solution presented at the end of the previous chapter. Using monotonic relationships to classify new design points simply shifts the error rates of the baseline process to the left because the same point sequence is used but fewer simulation runs are required. The efficiency gain is considerable.

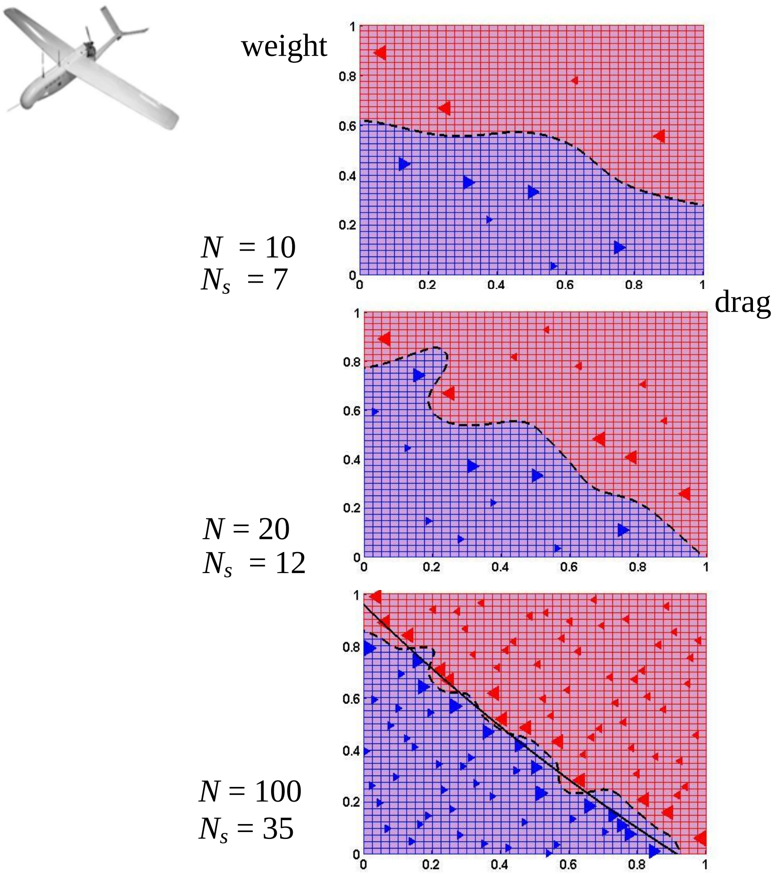


Figure 6.3. The Systems' Classifier for 10, 20, and 100 Training Points

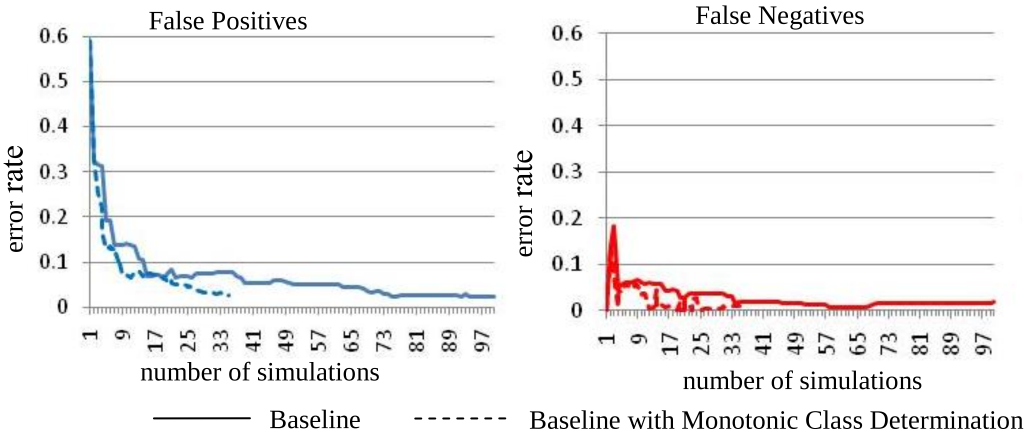


Figure 6.4. KBN Classifier Error Rates for the Systems' Design Space

6.5 DISCUSSION

Two simple means of improving the baseline classifier by exploiting knowledge of monotonic relationships were demonstrated in this chapter. In the first method, the training set was seeded with points of known class with all subsequent points being sampled and evaluated according to the baseline classifier solution. The seeded points provided a better decision boundary than the baseline solution for low numbers of sample points. In the second method, the efficiency of the baseline classifier solution was improved by providing the same error rates but at the cost of far fewer simulation runs. The first method of seeding the training set will be discussed next, followed by a discussion of the second method of determining the class without simulation.

The presented method of seeding the training set with heavily weighted design points of known class was demonstrated using monotonic reasoning. But the same procedure can be used to combine knowledge from multiple sources and not just from sampling a simulation or monotonic reasoning. However, the presented method of simply filling the training set with new points is limited in several ways. First, the class of the seeded points must be known with a high level of confidence or else the training set will be corrupted and the resulting classifier's performance could be significantly compromised. Second, although the weight of the seeded training points can be inflated by including replications of the design points in the training set, the influence of the seeded points on the decision boundary is more dependent upon the size of the standard deviation relative to the proximity to the decision boundary. This observation suggests that the seeded points should perhaps have a different standard deviation than the points sampled from the simulation. Future work can explore the impact of implementing this increased flexibility. Third, the deterministic sampling according to the Halton sequence did not account for the location of the seeded training points. Ideally, the seeding could occur anywhere in the design space and the subsequent sampling would adapt to not sample again in the same region as the seeded design points. The possibility of exploratory sampling in regions of low information is demonstrated in the next chapter.

The presented method of using monotonic reasoning to classify a new sample point relied upon all design parameters having a monotonic influence on the performance parameter. If the performance parameter is not monotonic with respect to all of the design parameters, then the performance gain would be much more limited because a Halton sequence purposefully does not repeat sampling at the same value for any dimension. For example, if the systems' design space was monotonic in weight but not drag then using monotonicity to classify a new point can only be guaranteed if it is dominated with respect to weight by an existing training point with the same drag. Thus, extending the results of this chapter to the more general case of monotonic relationships between any subset of the design parameters and the performance parameter would require a new sampling strategy that fills the training set with new points of known class in the appropriate directions along all of the dimensions that have a monotonic relationship. A demonstration is reserved for future work. As discussed in the previous paragraph, any subsequent sampling would also need to adapt to take into account the regions of known information and avoid sampling there. Adaptive exploratory sampling that achieves this avoidance of regions of high information is developed in the next chapter.

Finally, the decision boundaries generated using both of the methods in this chapter are not monotonic as is most evident in Fig. 6.1 and 6.3 for the case of 20 training points. The demonstrated methods do not enforce consistency of the decision boundary with the monotonicities. Future work could look at restricting the allowable values of standard deviations such that the decision boundary is consistent with the monotonicities of the problem. Future work could also develop new classifiers with decision boundaries that are consistent with the monotonic relationships of the problem.

Chapter 7. Adaptive Sampling

In previous chapters, classifiers have been based on data obtained from the deterministic space filling Halton sequence. The shortcoming of deterministic space filling sampling schemes is that they do not adapt to focus the samples on regions of interest to the design team (exploitation) or, alternatively, on regions in which little information is known (exploration). In this chapter, the KBN classifier is used to generate new sample locations that follow either of two strategies: exploration or exploitation. An exploratory sample is in a location of the design space of low sampling density, and an exploitive sample comes from a region of known acceptability. The goal of an exploratory sampling strategy is to find new and better regions of the design space. In contrast, an exploitive sampling strategy spends its resources in regions of known performance in order to find similar but better designs.

The primary goal for providing the capability of exploratory and exploitative sampling is to give the designer an enhanced ability to spend resources in a way that is more flexible than simple space filling non-adaptive sampling methods. The designer's search strategy will depend generally on the nature of the problem as well as the type and quantity of prior information that is available. Accordingly, different strategies will be discussed, but some effort will be focused on identifying a general strategy that works well for the example UAV design problem in terms of two benefits: 1) fewer resources are spent sampling the region of the subsystems team's design space that does not fall within the systems team's search domain and 2) a greater quantity of acceptable designs are identified but not at the expense of losing classification accuracy.

The next section introduces the novel KBN space filling exploratory sampling method including its use in conjunction with exploitive KBN sampling. Section 7.2

reviews other adaptive sampling methods and discusses their relationship to the proposed method. Section 7.3 compares the performance of the proposed KBN space filling method on the example UAV wing design problem to the baseline solution that uses a Halton sequence. Section 7.4 studies the use of more exploitive strategies for the UAV wing design problem. The last section discusses the results.

7.1 KBN CLASSIFIERS FOR EXPLORATORY SAMPLING

Because KBN classifiers use a probability distribution estimate for defining regions of the design space, exploitive sampling can draw directly from these distributions. This capability was an early motivator for the use of generative classifiers as discussed previously. However, there is not an explicitly defined probability distribution defining regions that have not yet been sampled. A less direct method will be required to generate exploratory samples.

The proposed method for determining the next exploratory sample point relies upon constructing a kernel density estimate, called the exploratory KDE, based upon all N of the design points from both the acceptable and unacceptable classes and finding its minimum. The exploratory KDE is multimodal by design, with local maxima at the already sampled design points and local minima at the design regions of the lowest sampling density. If the exploratory KDE's standard deviation is too small, large regions of the exploratory KDE will be near zero. If its standard deviation is too large, the minimum will always lie on the border of the search domain. However, if the standard deviation is neither too large nor too small, by the definition of the KDE, the global minimum will be the point that is the furthest from all of the training set designs and hence will be the best choice for the next exploratory sample, $\hat{\boldsymbol{x}}^{N+1}$, according to Eq. 7.1.

$$\widehat{\boldsymbol{x}}^{N+1} = argmin\left(\check{p}\left(\boldsymbol{x}, \widehat{\boldsymbol{x}}^{j=1\ldots N}, \check{\sigma}\right)\right) \text{ subject to } \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \tag{7.1}$$

The exploratory KDE has been designated $\check{p}$ and is a function of a design point, $\boldsymbol{x}$, all of the previously sampled design points, $\widehat{\boldsymbol{x}}^{j=1\ldots N}$, and the exploratory standard deviation, $\check{\sigma}$.

The key to the success of the exploratory KDE lies in setting the standard deviation, $\check{\sigma}$, to a good value. If the design points are spread out such that they fill the D-dimensional design space evenly then the smallest distance between a sample and its nearest neighboring samples can be approximated as the length, l, of the edge of a grid according to Eq. 7.2.

$$l = \frac{1}{N^{1/D} - 1} \tag{7.2}$$

Within this hypothetical grid, the exploratory KDE minima will occur at the center of a line segment in 1 dimension, an area in 2 dimensions, a cube in 3 dimensions, etc, that are defined by the training point locations on the grid. The minima will have 2^D nearest neighbors that are at a distance of $\frac{l}{2}$ in each of the D dimensional directions. Assuming that only the 2^D nearest neighboring points have a significant contribution to the probability at the minima, then Eq. 7.3 can be used to approximate the minima of the exploratory KDE.

$$p_* \cong \frac{1}{N} \sum_{j=1}^{2^D} \prod_{i=1}^{D} \frac{1}{\check{\sigma}\sqrt{2\pi}} e^{-\frac{\left(\frac{l}{2}\right)^2}{2\check{\sigma}^2}} = \frac{1}{N} \frac{2^D}{\check{\sigma}^D (2\pi)^{D/2}} e^{-\frac{Dl^2}{8\check{\sigma}^2}} \tag{7.3}$$

It is simpler to not normalize the Normal distributions or to divide by the number of sample points, scaling Eq. 7.3 to Eq. 7.4 without moving the minima.

$$p_* \cong 2^D e^{-\frac{Dl^2}{8\check{\sigma}^2}} \tag{7.4}$$

Assuming that none of the *2D* adjacent design points has a significant influence on the value of the exploratory KDE at a training point, and that the kernel is neither normalized nor averaged, the local maxima will be approximately 1. Setting the value at the minima of the exploratory KDE to be half of the value of the maxima, according to Eq. 7.5, a formula for the standard deviation can be derived following Eq. 7.6-7.9 that is a heuristic for providing well defined minima in between the training points.

$$2^{D} e^{-\frac{Dl^2}{8\breve{\sigma}^2}} = \frac{1}{2} \tag{7.5}$$

$$e^{-\frac{Dl^2}{8\breve{\sigma}^2}} = \frac{1}{2^{D+1}} \tag{7.6}$$

$$\frac{Dl^2}{8\breve{\sigma}^2} = (D+1)ln(2) \tag{7.7}$$

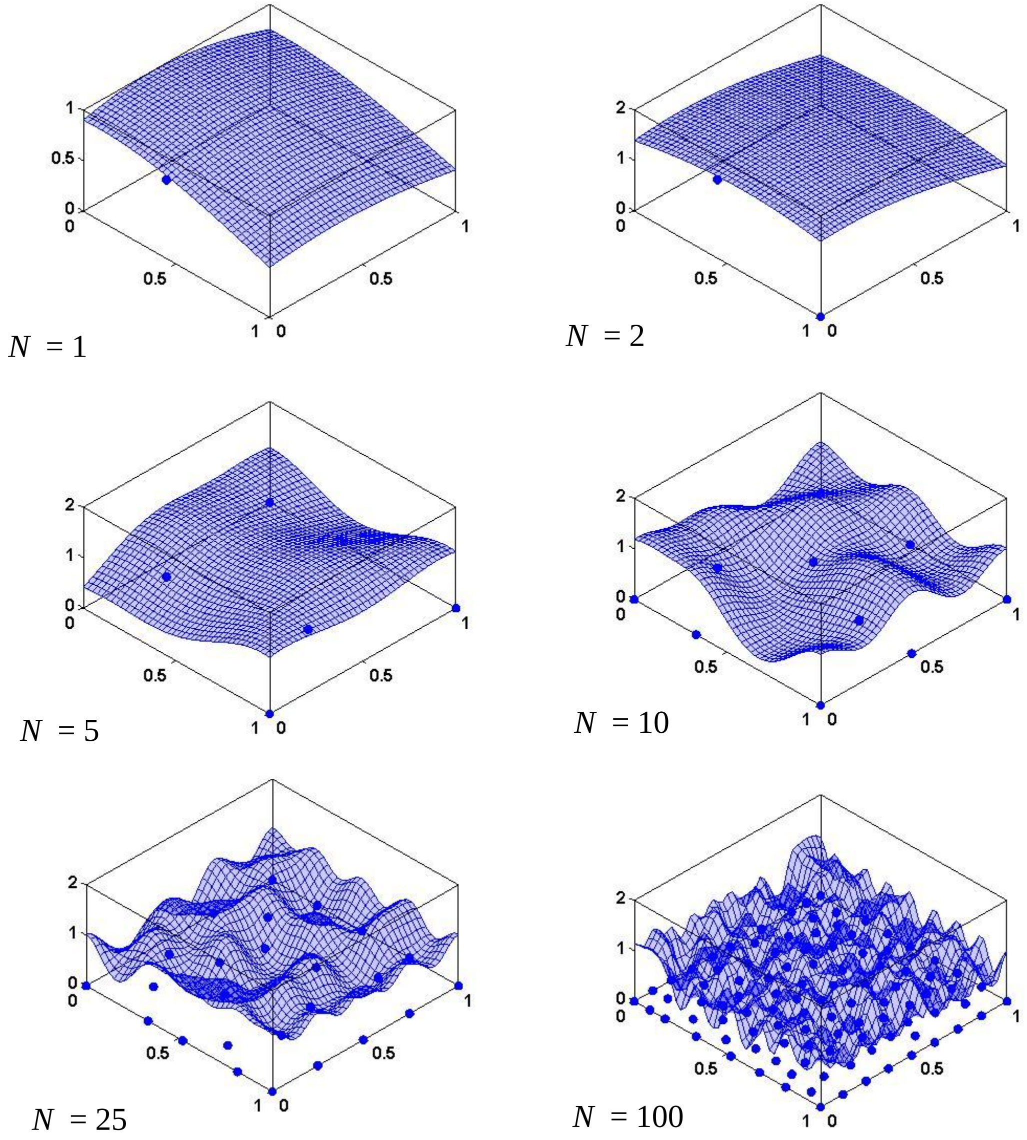
$$\breve{\sigma} = \left(\frac{1}{8ln(2)}\right)^{1/2} l \left(\frac{D}{D+1}\right)^{1/2} \tag{7.8}$$

$$\breve{\sigma} = \left(\frac{1}{8ln(2)}\right)^{1/2} \left(\frac{1}{N^{1/D}-1}\right) \left(\frac{D}{D+1}\right)^{1/2} \cong 0.425 \left(\frac{1}{N^{1/D}-1}\right) \left(\frac{D}{D+1}\right)^{1/2} \tag{7.9}$$

Equation 7.9 provides an approximate formula for calculating the standard deviation that will produce well defined local minima that are roughly half the height of the local maxima at the training points laid out in a grid. The exploratory standard deviation scales primarily by the number of sample points in a manner that is similar to the empirical formula previously used with two dimensions. The usefulness of this equation will be demonstrated for two dimensions although the derivation might lead to effective results in other dimensions too. If the exploratory standard deviation is too large the minimum will repeatedly fall in the corners of the design space. If the exploratory standard deviation is too small the minimum will be found to be near zero. Both of these possibilities can be detected and corrected for in a more robust future implementation.

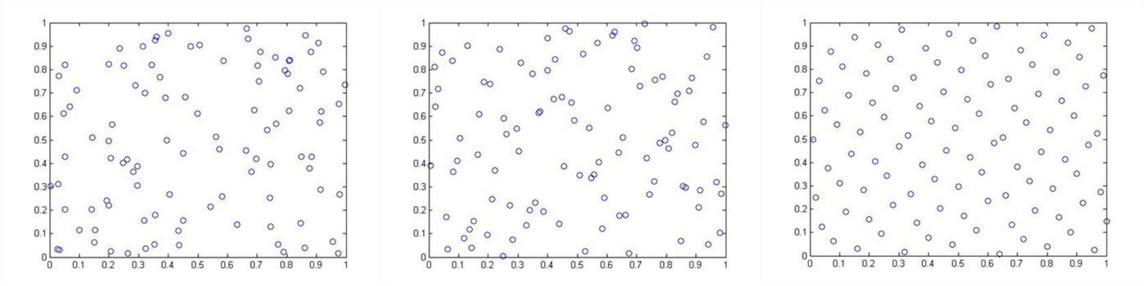
Using Eq. 7.9 to set the standard deviation and randomly choosing the first point, the exploratory KDE's for 1, 2, 5, 10, 25, and 100 sequentially sampled space filling training points are shown in Fig. 7.1. The next point was chosen as the minimum of the KDE according to Eq. 7.1 and as determined by a sequential quadratic programming (SQP) optimization from random starting points using Matlab®'s *fmincon* function. The search was stopped after either 0.1 seconds was reached or five consecutive SQP runs failed to lower the best minimum by more than 1%. It should be noted that the derivatives are available for the optimization. The Matlab® software implementing this procedure is in Appendix F.

A space-filling pattern for the proposed method is compared in Fig. 7.2 to other popular space-filling designs: random, Latin hypercube, Hamersley, and Halton sequences. The proposed method produces a very well dispersed pattern that is similar to a grid except with many more levels captured for each variable. Also, relative to the other sequences, the proposed method places samples on the search domain boundary which can be considered good or bad depending on the problem. The lower right graph of Fig. 7.2 demonstrates the proposed method's ability to produce exploratory samples that avoid the twenty previously sampled normally distributed solid points and to fill in the space around it—the desired behavior that motivated the creation of the proposed method.



$N = 1$ $N = 2$ $N = 5$ $N = 10$ $N = 25$ $N = 100$

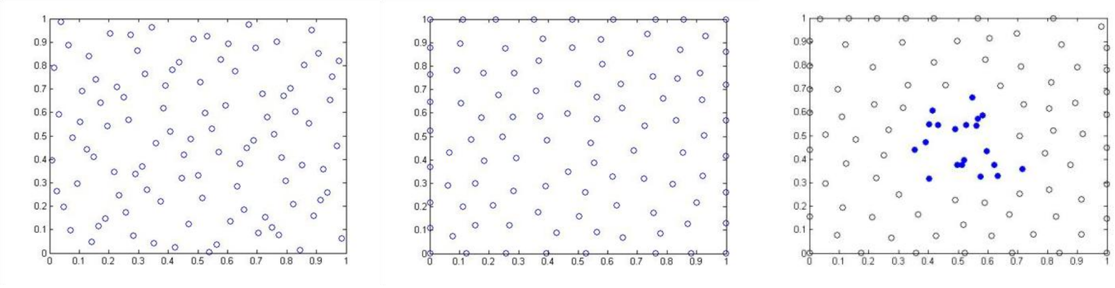
Figure 7.1 Exploratory Samples for $N = 1, 2, 5, 10, 25,$ and 100 Points



random

Latin hypercube

Hammersley



Halton

KBN exploratory

KBN exploitive
and exploratory

Figure 7.2 Different Methods for Producing Space-Filling Samples

The demonstrated effectiveness of the mixed exploitive/exploratory sampling with non-uniformly dispersed training points does not invalidate the use of the standard deviation that was earlier derived from the assumption that the training points lie on a grid. The method is hence not particularly sensitive to the choice in standard deviation and the heuristic approach used to set it appears to be effective. However, more tests need to be conducted to fully validate the approach taken to set the standard deviation. As mentioned earlier, adjustments might need to be made if samples repeatedly fall in the search domain corners or the minima are very near zero. The next section applies the KBN space-filling sequence to the example UAV wing design problem.

7.2 BACKGROUND

At the heart of the proposed adaptive sampling methods is the ability to flexibly blend exploitive and exploratory sampling strategies. Stochastic search methods such as simulated annealing (Kirkpatrick et al., 1983; Cerny, 1985) feature a similar capability of changing search strategies between exploration and exploitation through a cooling schedule. Genetic algorithms (GA's) typically begin with a randomly generated exploratory population from which operators select and construct the next generation of points (Holland, 1992; Goldberg, 1989). The basic GA operators include a selection step for exploiting good solutions as well as crossover and mutation steps that look for new solutions, although how these operators search in terms of exploration/exploitation is not precisely clear (Eiben and Schippers, 1998). More recent developments suggest that adapting the GA strategy throughout the search can produce better results with a modest increase in computational cost (Hanna and Cagan, 2010).

The sequential sampling literature often exploits the information from a metamodel to direct the search toward a variety of goals including optimal regions and unexplored regions. Sasena (2002) proposes a strategy that switches between exploitation and exploration, and Turner et al. (2007) propose a multiobjective formulation that blends four different sampling goals. One of the strengths of the proposed adaptive search methods is the ability to switch between exploratory and exploitive search strategies. However, this capability is not extensively explored in this chapter. Instead, two extreme cases are considered: purely exploratory and aggressively exploitive. As will be seen, even the aggressively exploitive strategy relies on a minimum of initial exploratory sampling to avoid sacrificing classification accuracy. An interesting line of future research could attempt to identify the best mixed exploratory/exploitive sampling strategy for different categories of test problems.

The proposed exploitive sampling strategy is a form of stochastic search in which a probability distribution is constructed to define a region of the design space which can be directly sampled for more points within that region. The basis for the proposed exploitive sampling is the class of optimization methods called estimation of density algorithms (EDA's). The continuous variable iterated density estimation evolutionary algorithm (ID𝔼A) uses a single KBN constructed with the top *M* performing design points and samples it for the next generation (Bosman and Thierens, 2000). These algorithms are aligned with the genetic algorithm paradigm but with a more explicit definition of where the better points are likely to be. Similar to GA's, the IDEA framework maintains a constant population size for every generation, discarding underperforming designs as needed. In contrast, this research exploits the knowledge of all of the acceptable design points. A design point that satisfies acceptability constraints will remain acceptable while a design point that is within the current best set will not necessarily remain among the best. By pursuing acceptable design regions instead of optimal regions, there is less risk of over-committing resources to a local minimum. However, there is a greater computational expense in using an ever-expanding database. A memory-bounded version of the presented methods would be an interesting direction for future work. Additionally, defining another classifier in terms of the best performing points or weighted according to the objective function would add even more flexibility in the decision of where to look next in the design space.

EDA's typically rely upon a large initial random sample to explore the design space after which the sampling is exclusively exploitive. In contrast, the proposed exploratory method was designed to allow for exploratory sampling at any stage during the search process. Because the KDE can be built on any set of data points, an exploratory sample can be generated at any time during the search process. This adaptability sets the proposed exploratory sampling mechanism apart from popular space-filling sampling methods such as Latin hypercube sampling (Mckay et al., 1979) and Halton sequences (Halton, 1960).

The literature on sequential sampling is full of adaptive space-filling sampling methods that work in conjunction with metamodels (for example: Koehler and Owen, 1996; Jones, 2001; Jin et al., 2002; Sasena, 2002; Turner et al., 2007). The proposed method does not rely upon information obtained from metamodels and the associated costs of training them, and hence this review does not cover these and derivative methods any further. Nevertheless, some methods developed in conjunction with metamodeling are included in the next review because of the possibility of using them without a metamodel. Adaptive exploratory sampling methods that do not use metamodel information, including the proposed method, often take a similar, direct approach: they search for the point that is the furthest from the existing points. These methods include Maximin (Johnson et al., 1990), qualitative and quantitative sequential sampling (Q2S2) (Rai and Campbell, 2008), the proximity-based method proposed for use in conjunction with NURBs-based metamodels (Turner et al., 2007), and minimal Kullback-Liebler information designs (Jourdan and Franco, 2009). These methods are reviewed next.

The maximin approach seeks a point set that maximizes the minimum distance between any two given points. This method was initially presented as a batch process where the locations of all points in the next sample set are placed in the design space simultaneously (Johnson et al. 1990). Implementing such a design could be costly because the number of variables in the optimization could be high. A more practical implementation is to use a batch size of one and to sequentially place each point such that the minimum distance between the new point and all of the existing training points is maximized (Jin et al., 2001). This should produce a very similar result to the proposed method although the implementation is slightly different in that no attempt is made at identifying the closest point in the proposed approach; all points contribute to the distance measure although often by a small amount. For example, if two points were coincident in the KBN training set then they would both contribute to the density estimate used to place the next point whereas the maximin approach would not account for the presence of both points. However, for the proposed method the effect of a point on the density diminishes exponentially with the distance and the standard deviation is purposefully set low, so the difference is slight.

In Q2S2, a confidence surface is constructed as a factored product of a kernel function in each dimension with an additional product for the performance parameter that produces a 1 at a data point denoting that this point is known with complete confidence and that decays to -1 along the performance parameter axis denoting that these other values for the output are, with complete confidence, not associated with the design point. Along the design parameter axes, the confidence value decays to 0 with increasing distance from a data point, denoting that less information is known about the performance of the design space as one moves further from the design point. The confidence function is integrated along the performance parameter axis, combined with all other integrated confidence fields from the other design points, squared, and minimized in order to find the next design point. The proposed exploratory sampling method is similar in nature to the Q2S2 process. However, the proposed method does not use the additional performance parameter axis and the associated integration and squaring. Hence, the proposed method should produce similar results but at a lower computational cost. Furthermore, the standard deviation of the Gaussian kernels is specifically controlled in the proposed method to produce well defined minima. However, Q2S2, as discussed previously, has the attractive capabilities of accounting for variable fidelity sources that are not present in the current implementation of the proposed method.

For space-filling sampling as proposed by Turner et al. a normalized proximity function is constructed as a tensor product of parabolic spans between the nearest control points of the NURBS surface (Turner et al., 2007). If the control points coincide with the data points, then this method should produce very similar results as the proposed approach. Furthermore, by controlling the location of the control points such that the density of control points is correlated to the data point density then the control points provide a reduction in computational complexity. However, using the control points to define the proximity minima might oversimplify the exploratory search. Further study is required to fully understand the potential advantages of proximity-based sampling using the control points of NURBs-based metamodels versus the proposed approach. Some form of clustering will be necessary at high numbers of samples in order to reduce the computational complexity of the proposed exploratory search.

A maximum entropy sampling approach that is not for use in conjunction with metamodels was recently proposed by Jourdan and Franco (2009) called minimum Kullback-Liebler (KL) information designs. The method is based upon an estimation of the entropy of a Gaussian kernel distribution. Monte Carlo sampling or a nearest neighbor distance approximation are used to estimate the entropy that is used in an exchange algorithm for finding the best space-filling distribution of points. Estimating the entropy will be a very expensive approach, but for the cases when it is practical to use the minimum KL information designs the results will also be similar to the proposed method. There is however a subtle difference. The proposed method is more likely to place points on the search domain's boundary than the minimum KL information designs because the entropy of a KDE measured over a fixed search domain will be slightly higher if the exterior points lie just off the edge (the KDE is more approximately a uniform distribution over the search domain). However, the sequential nature of the proposed method in addition to not estimating the entropy but instead performing a limited number of gradient-based searches for which the derivatives are available, will be substantially less computationally expensive.

7.3 UAV SOLUTION 6: EXPLORATORY SAMPLING

In this section, the UAV wing design problem is solved again except this time the points are not generated from a Halton sequence but come from the exploratory sampling procedure developed in the previous section. Because the KBN exploratory sampling

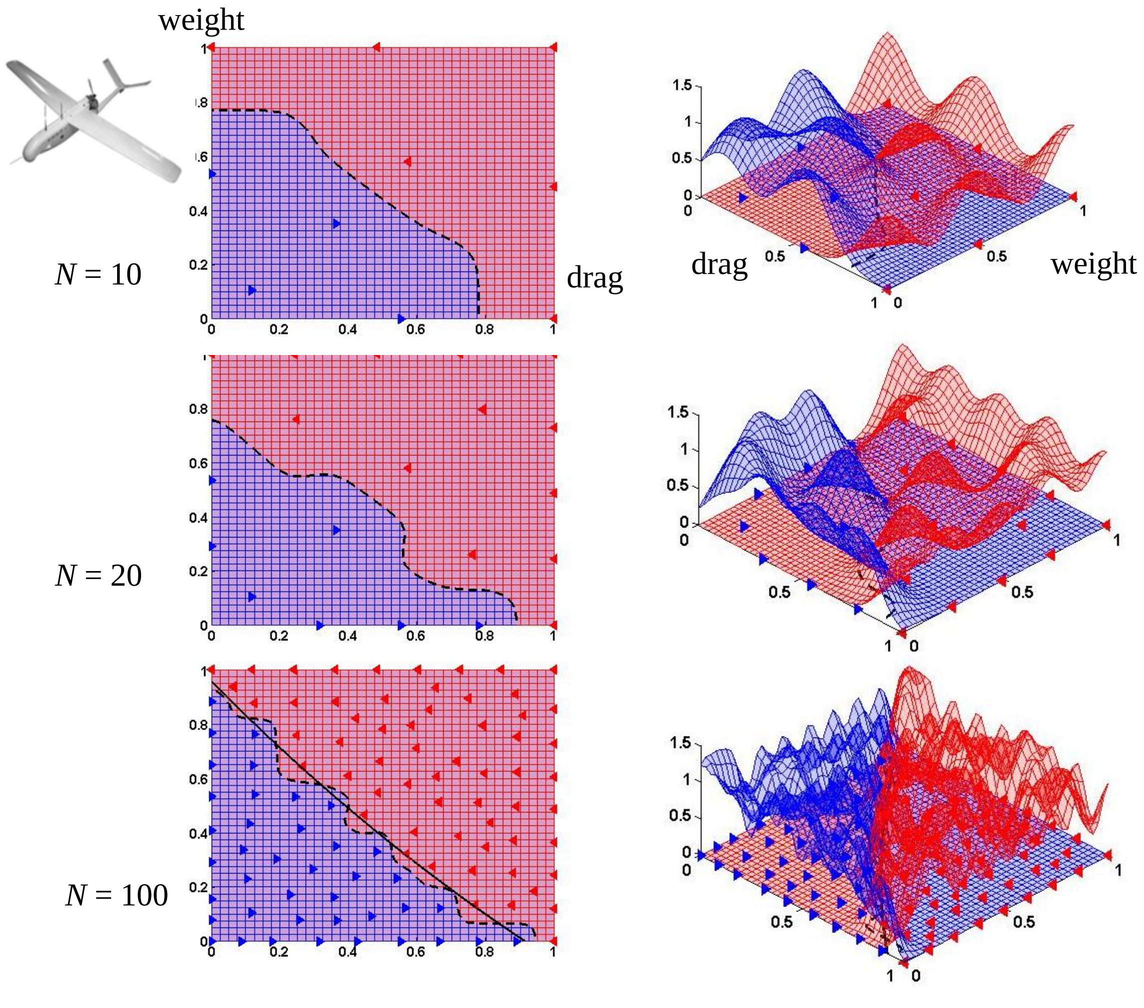


Figure 7.3 The Systems' Classifier for 10, 20, and 100 Training Points

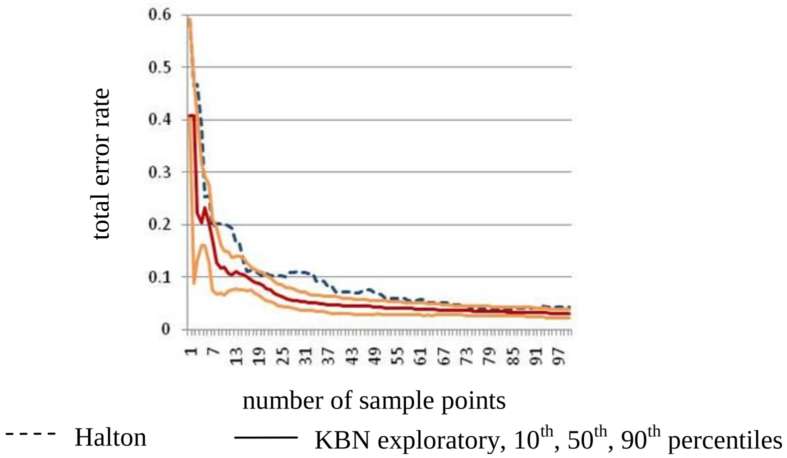


Figure 7.4 KBN Classifier Total Error Rates for the Systems' Design Space

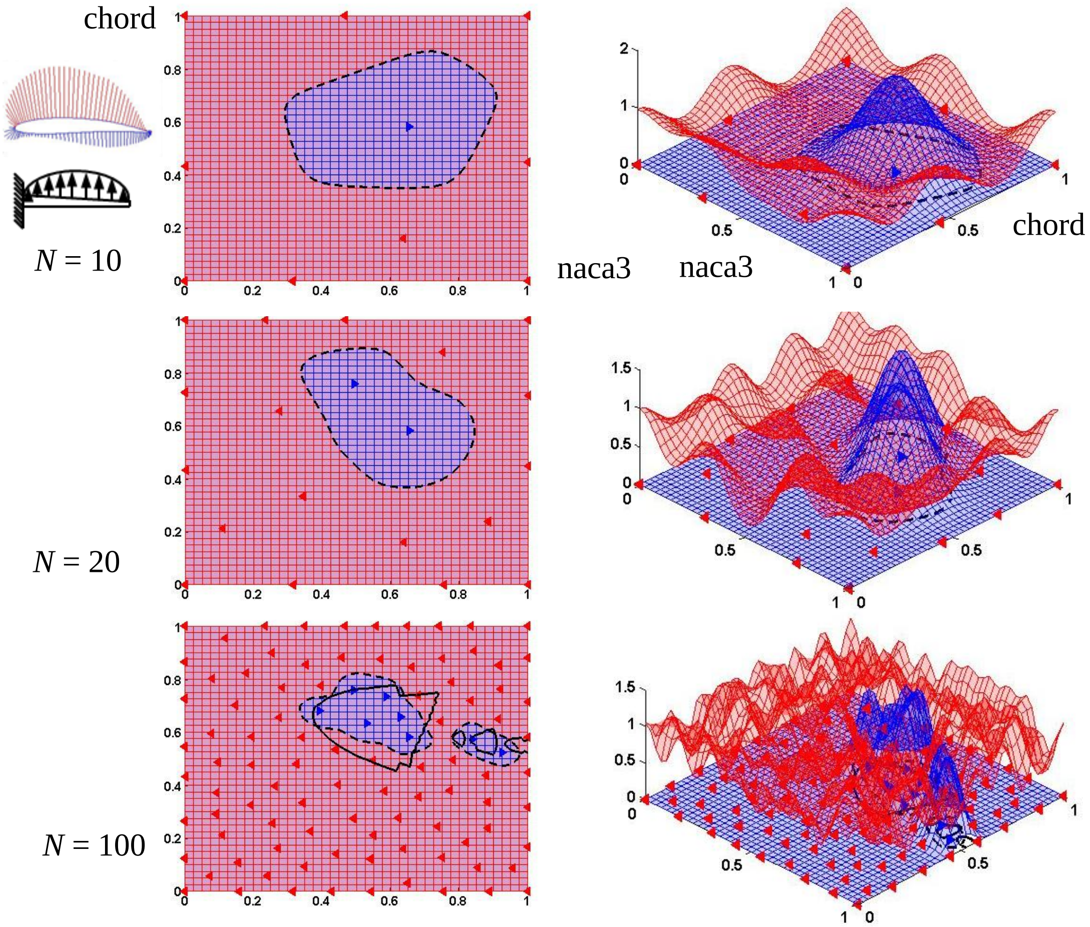


Figure 7.5 The Subsystems' Classifier for 10, 20, and 100 Training Points

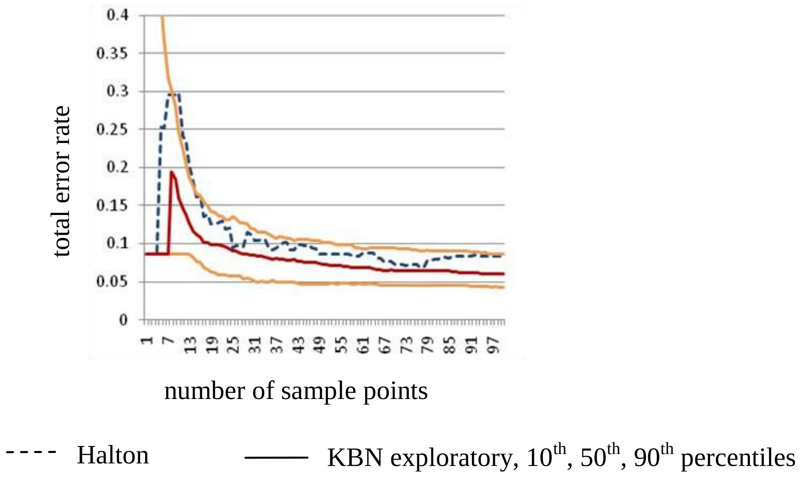


Figure 7.6 KBN Classifier Total Error Rates for the Subsystems

sequence is random, the solution is repeated and the 10th, 50th, and 90th percentile total error rates that are the sum of the false positive and false negative error rates are reported. The experiment was repeated in batches of 100 runs until the reported percentiles did not change by a total error rate of more than .01 relative to the results from the previous batches. Figure 7.3 shows the results from a single run for the system's design problem for 10, 20, and 100 KBN exploratory training points. The total error rates are reported in Fig. 7.4 for the systems level and compared to the baseline solution from the end of Chapter 5 that used a Halton sequence. The subsystem results are presented in Fig. 7.5 and 7.6.

Generating exploratory samples by finding local minima of KBN's can produce acceptably low error rates at both the systems and subsystems levels. Furthermore, the error rates are robust to the randomness in the sampling sequence. An interesting difference between the Halton sequence and the exploratory KBN sequence is that the former does not place points on the search domain boundary. As a result, the Halton sequence will have a slightly higher density of sampling at the interior of the design space, providing a slight advantage for the subsystems problem where many of the points near the border are either too heavy or have too much drag to be within the region searched at the systems level. This difference is illustrated in Fig. 7.7 where the number of subsystem design points that are within the systems' search domain is plotted as a function of the number of training points. There is a simple potential remedy for the KBN exploratory sequence: artificially restrict the exploratory sample domain to a smaller rectangular region. However, a more general approach is considered in the next section: use another classifier to find the boundary that defines the region that maps to the systems' domain.

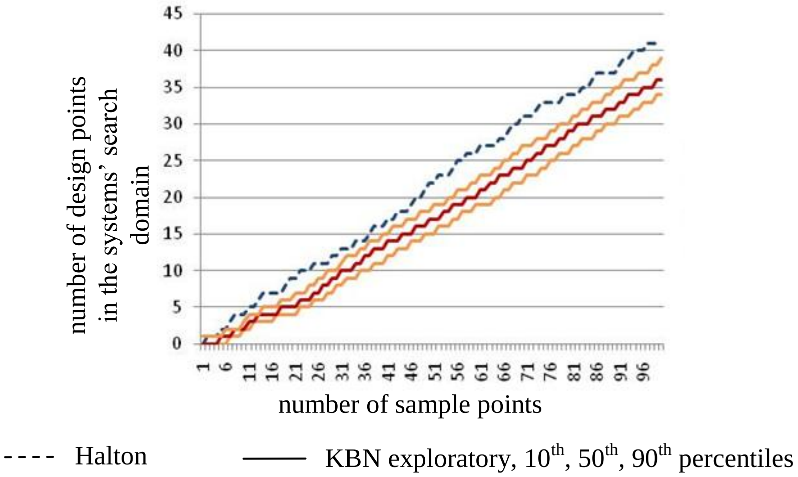


Figure 7.7 Subsystems' Design Points within the Systems' Search Domain

7.4 UAV SOLUTIONS 7 AND 8: EXPLORATORY AND EXPLOITIVE SAMPLING

In this section progressively more aggressive exploitive sampling strategies are pursued as solutions to the example UAV wing design problem and compared to the purely exploratory sampling strategy of the previous section. The comparison is in terms of both classification error rates as well as the number of subsystem design points that are within the systems' search domain. The goal is to raise the latter without compromising the former.

The first study in this section introduces another classifier that is trained to identify the regions of the subsystem design space that have both low enough drag and weight to be within the rectangular bounds of the systems' search space. Every sample point is classified as being acceptable if it is within the systems' search domain and unacceptable otherwise. The new search domain classifier for the subsystem design space is illustrated in Fig. 7.8 for a KBN exploratory sampling sequence of 100 training points. The correct decision boundary is also shown: above the top boundary are the designs with a normalized drag greater than one, and below the bottom boundary are the designs with a normalized weight greater than one. All points in between these boundaries are within the systems' rectangular search domain. A large portion of the subsystem design space is not worth exploring because either the weight or the drag is simply too high.

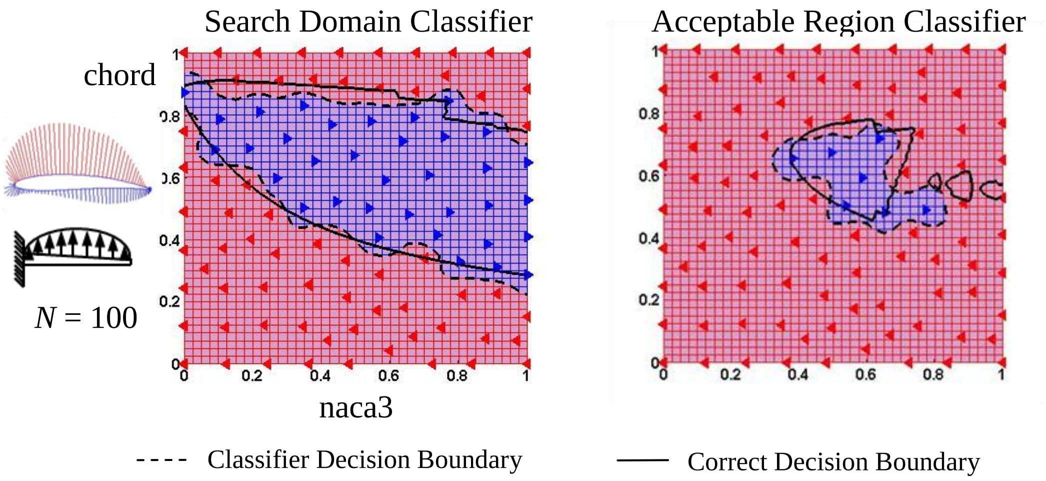


Figure 7.8 Subsystems' Search Domain and Acceptable Region Classifiers

The first mixed exploratory/exploitive sampling strategy that is investigated explores the design space until a crude estimate of the search domain classifier exists after which its acceptable region is sampled exclusively. At least ten and if necessary more KBN exploratory design points are sampled until the first design is found that falls within the systems' search domain, after which the search domain classifier's acceptable

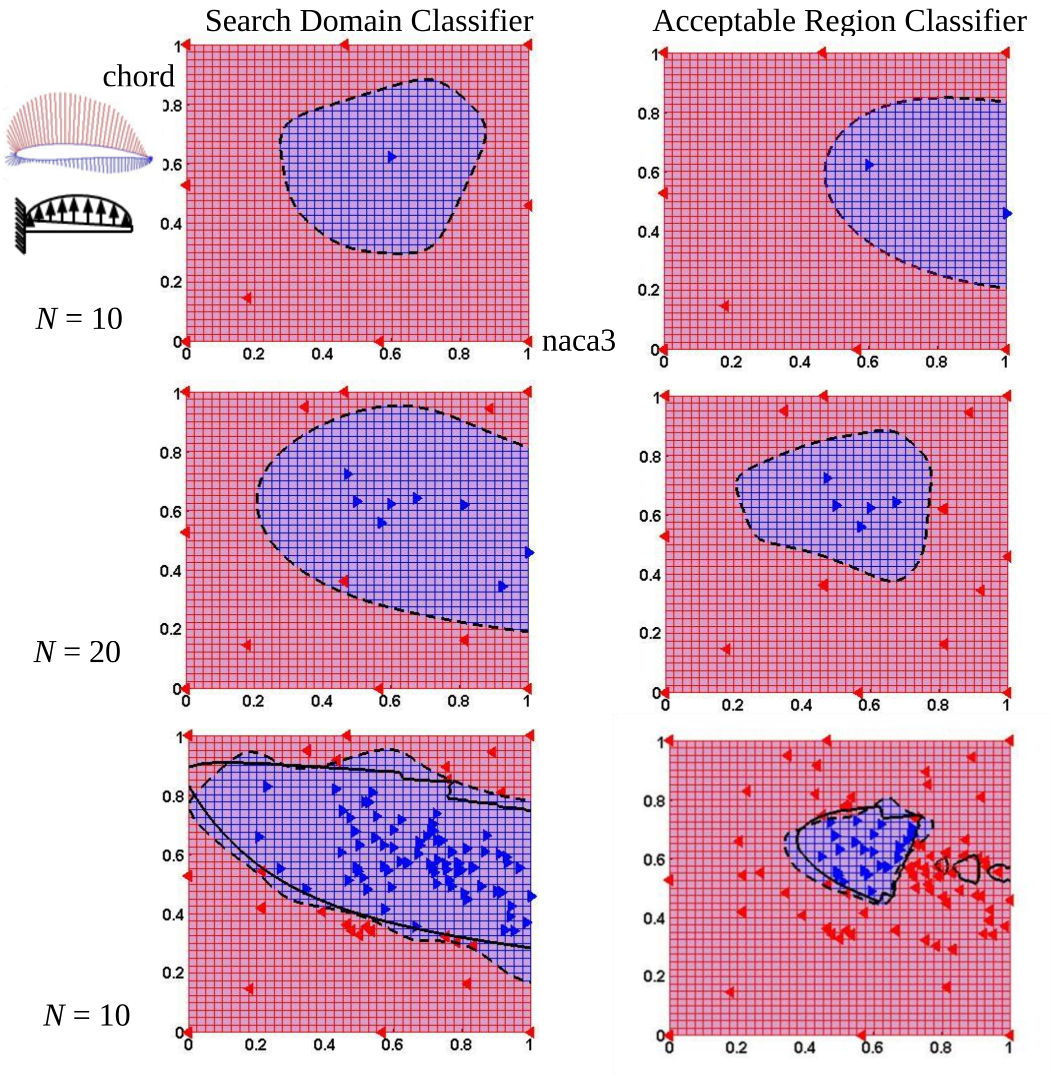


Figure 7.9 Subsystems' Search Domain and Acceptable Region Classifiers

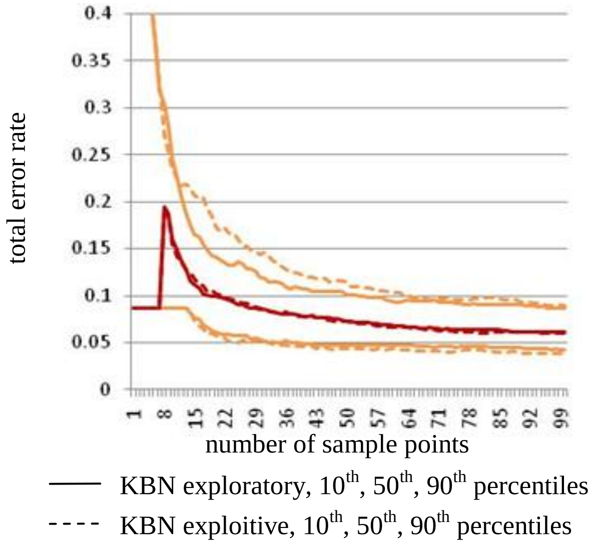


Figure 7.10 KBN Classifier Total Error Rates for the Subsystems

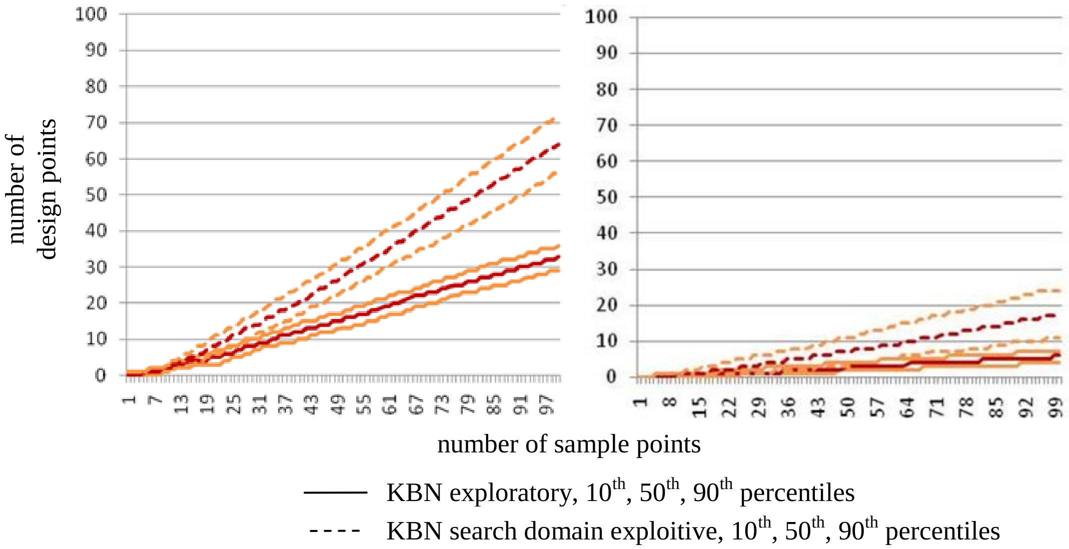
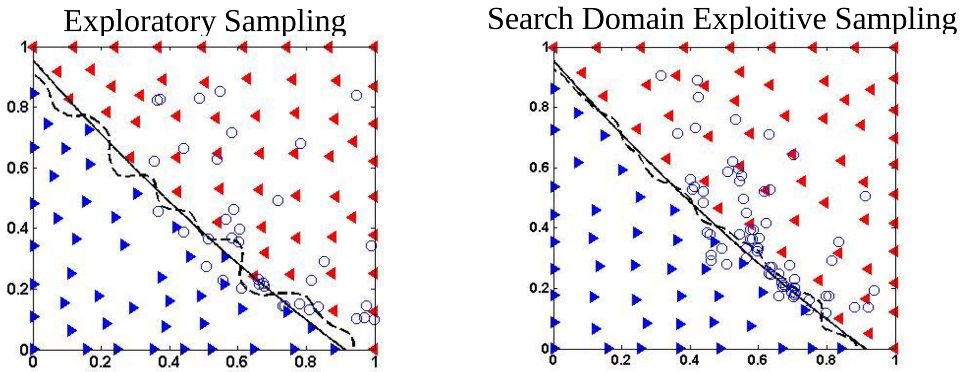


Figure 7.11 Subsystems' Design Points within the Systems' Search Domain (left) and within the Systems' Acceptable Region (right)

region is sampled exclusively until 100 total training points have been generated. Because the exploitive samples should be well dispersed throughout the design space, the search domain classifier's standard deviations are calculated according to Eq. 5.11 but with a numerator equal to 1. By increasing the standard deviation, the samples will be less likely to clump around the initial points, providing a more uniform sampling over the desired region. The increased standard deviation will also mean that the search domain classifier will misclassify some of the training points as discussed in detail in Chapter 5. But this is not so important when the goal is to uniformly sample a region and not to classify it which was the focus of the studies in Chapter 5. The possibility of an exploitive sample falling outside of the desired region might even better define the region's boundary. However, future work should more rigorously identify how to set the standard deviation such that it reliably produces a uniform sample over the desired region.

Representative classifiers for 10, 20, and 100 training points are shown in Fig. 7.9, and the 10th, 50th and 90th percentile total error rates are compared in Fig. 7.10 to the purely space-filling strategies using the exploratory KBN. Figure 7.11 shows the number of design points within the systems' search domain for the two strategies as well as the number of design points within the systems' acceptable region.

From these results, it is clear that the search domain exploitive sampling strategy accomplishes the goal of increasing the number of subsystem level design points in the systems' search domain without compromising the classification error rates for the mutually acceptable design region classifier. Sampling the search domain classifier increases the probability of a sample lying within the systems' search domain and also generates samples of both the acceptable and unacceptable classes for defining the decision boundary of the mutually acceptable design region. Fig. 7.12 shows the resulting sampled design points from a subsystem search domain exploitive strategy on the left and a purely exploratory strategy on the right mapped onto the systems' design space as the hollow circles. The plots clearly illustrate the advantage of the exploitive strategy in terms of generating more points within the systems' search domain as well as more acceptable design points being identified. However, the subsystems' search domain exploitive strategy does not disperse the points throughout the systems' design space as well as the exploratory strategy which provides a slightly more thorough sense of what designs are achievable by the subsystems' group. The tradeoff is as expected: a more exploitive strategy achieves a higher density of similar high performance designs while a more exploratory strategy achieves a lower density of more diverse designs.


Exploratory Sampling

Search Domain Exploitive Sampling

Figure 7.12 Subsystems' Design Points Mapped to the Systems' Design Space

The second strategy investigated in this section extends the results of the previous search domain sampling strategy to a more aggressive level of exploitive sampling by drawing samples exclusively from the design region that is acceptable to the systems group as soon as it is identified. The initial samples are exploratory for at least ten samples and continue to be exploratory until either the first point within the systems' search domain or the first point within the systems' acceptable region is found, after which the sampling is purely exploitive of the classifier with the first positive result. If the first positive sample is within the systems' search domain but not within the systems' acceptable design space, then the search domain classifier is sampled exploitively until the first acceptable design point is found. As soon as the first acceptable design point is found, its design space is sampled exploitively until a total of 100 training points have been sampled. The results for this more exploitive strategy are shown in Fig.7.13-17.

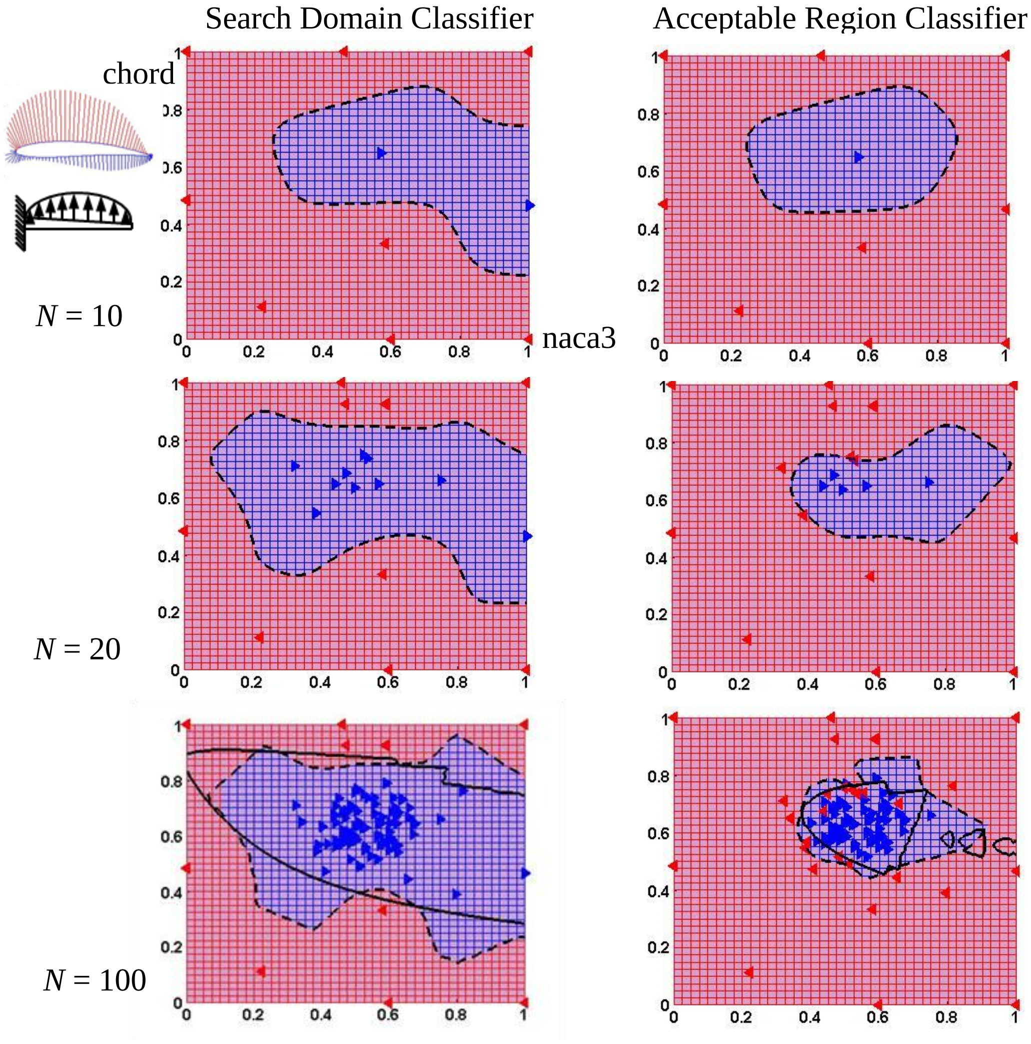


Figure 7.13 Subsystems' Search Domain and Acceptable Region Classifiers

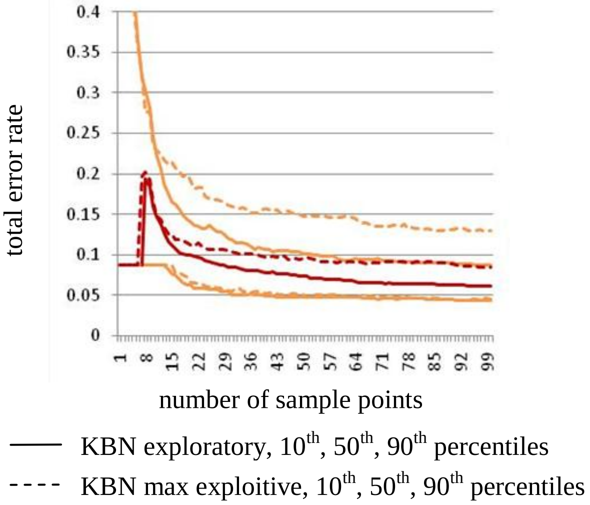


Figure 7.14 KBN Classifier Total Error Rates for the Subsystems

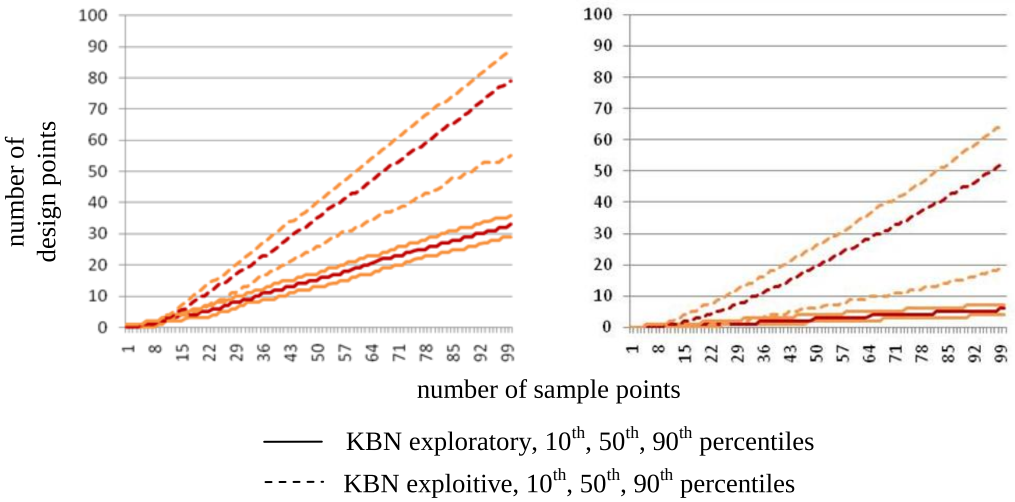
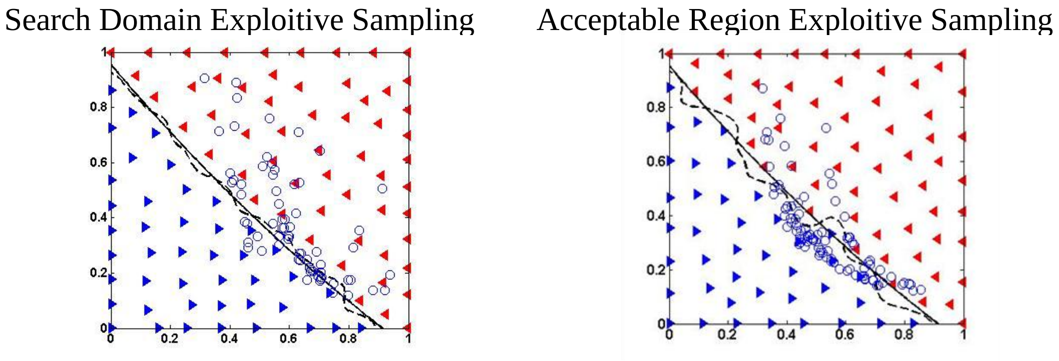


Figure 7.15 Subsystems' Design Points within the Systems' Search Domain (left) and within the Systems' Acceptable Region (right)


Search Domain Exploitive Sampling

Acceptable Region Exploitive Sampling

Figure 7.16 Subsystems' Design Points Mapped to the Systems' Design Space

From these results it is clear that an acceptable region exploitive sampling strategy, while producing the most points within the systems' acceptable region, can compromise the classification error rates. Part of the increased error might be explained by the rule-based standard deviation no longer being appropriate for the higher density of points concentrated within the systems' acceptable region. The lower right plot of Fig. 7.13 shows several misclassified unacceptable design points because of their proximity to the edge of the acceptable region. This might be remedied by adaptively setting the standard deviations using the method developed in Chapter 5. However, some of the red training points are also within the correct feasible region boundary as depicted by the solid line, implying that for this case the systems-level classifier misclassified some of the design points at the subsystems-level. When this occurs, using the adaptive method for setting the standard deviation from Chapter 5 could lead to worse classifiers. Future work should identify these borderline cases, assign them lower classification confidence levels, and take this information into account when constructing the classifier.

The other observation of immediate note is the large variation in the number of points found in the acceptable design region for the acceptable region exploitive strategy. From the right plot of Fig. 7.15, there appear to be a significant number of cases where the acceptable region is misclassified to such an extent that the samples taken from the classifier are not really acceptable to the systems group. This is most likely the result of trusting the classifier too soon, when only one acceptable point has been identified. Perhaps a more optimal strategy would spend some time exploring more in order to better define the acceptable region before it is sampled exclusively. The next section continues the discussion of the different sampling strategies.

7.5 DISCUSSION

In this chapter, using the KBN classifier for both exploitive and exploratory adaptive sampling was demonstrated. The ability to sample the probability distribution estimate of a region of the design space motivates the use of generative classifiers as opposed to discriminant approaches that do not create probability distributions. The density estimate, if carefully constructed, can also be used to find a sample point in a low density region of the design space for exploration. While these capabilities were developed and demonstrated in this chapter, there remains a lot of work to refine the methods. Some potential directions for refinement are discussed in this section.

Three simple sampling strategies were demonstrated in this chapter: purely exploratory, search domain exploitive and acceptable region exploitive. However, the exploratory and exploitive sampling methods can be combined into any general sampling strategy. Perhaps a mixed strategy is the best balance between exploit and explore such as a simulated annealing cooling strategy that transitions from exploration to exploitation. Perhaps a strategy that alternates between periods of exploration and exploitation would be better. It should be pointed out that as long as the probability of exploring is not zero, then eventually the search will find the global optimum, although this might involve an unreasonably high number of sample points. Critical to the problem of finding a generally applicable sampling strategy is the availability of test problems. Since the application is mechanical engineering design of complex systems, a variety of test problems from this domain should be used. Collaboration with industry would be extremely helpful in this regard. Even before additional test problems are evaluated, there are some important ways that the proposed methods can be investigated and improved.

The exploratory sampling method was demonstrated in this chapter for just two dimensional design spaces and its performance at higher dimensions will need to be studied to ensure its success. In particular, the calculation of the standard deviation has to be verified as being effective at higher dimensions. Furthermore, preliminary studies show that the method tends to place the design points on the edge of the design space where the density will naturally be low. At higher dimensions, this may be particularly undesirable if the acceptable region lies on the interior of the design space. Some other method of exploration may be necessary for finding low density but relatively nearby design regions. Furthermore, when either the search domain or the acceptable region was sampled to find more points within the same class, a higher standard deviation was used in order to avoid clumping of the new design points too closely to the existing training points within that class. The value for the standard deviation used for sampling was set empirically and more work needs to be done to develop a generally applicable rule for how the standard deviation should be set during the exploitive sampling. Finally, while the low impact of the exploitive sampling strategies on the classification errors demonstrated in this chapter suggest that the classifiers are relatively insensitive to the calculation of the prior class probability and standard deviation, the methods for setting these parameters could be revisited in order to produce even lower classification error rates in light of the less uniform distribution of the training points that results from exploitive sampling strategies.

The adaptive sampling strategies demonstrated in this chapter were sequential with the subsystems teams using a fully developed systems classifier for directing their sampling. When the sampling is adaptive and concurrent, there is a risk that design points will be placed in erroneously classified mutually feasible regions leading to a less efficient use of simulation time. An important next step for demonstrating the ability of these methods to meet the goals of this research is to test their effectiveness when used concurrently and to develop new sampling strategies that can manage this risk. For example, error and convergence measures can indicate when a classifier is ready to be used to guide the sampling process.

This concludes the development of the KBN classifier for the purposes of sharing locally mapped acceptable regions of the design space. More development along all of the lines mentioned throughout the previous chapters is warranted for further improvement of the classifier's ability to meet the perceived needs of a designer using the classifier in support of their collaborative design activity. However, further improvement upon the classifier is reserved for future work and the attention is now turned toward sharing the classifiers within more complicated networks of collaborating designers.

Chapter 8. Cross-Classification and Collaborative Classification Networks

In previous chapters a very simple collaborative relationship has been presented in which a subsystem design team's results serve as input to the simulations of the systems group. However, recall that the original UAV wing distributed design problem involved two subsystems teams: aerodynamics and structures. In previous solutions, these two subsystems teams were effectively merged by coordinating their design space sampling. The primary goal of this chapter is to develop and demonstrate a method for using KBN classifiers to separate the design activities of the two subsystems teams while maintaining agreement of their classifiers. The aerodynamics and structures teams share common design variables, *naca3* and *chord*. These two teams are also indirectly coupled through the systems' simulation because the structures group's wing weight affects the maximum allowable aerodynamics group's wing drag needed to achieve the necessary 900 km range. The implications of these interactions are discussed in depth in this chapter leading to a preliminary discussion of the interactions that this research can address and the ones it cannot.

To facilitate discussing interactions between collaborating groups of designers, collaborative classification network (CCN) diagrams, are developed in the next section. Section 8.2 presents cross-classification for the purposes of resolving subsystems that are coupled through a system-level dependency. Section 8.3 applies cross-classification to the UAV wing design problem and presents the results. Section 8.4 extends the CCN discussion to address dependencies that are not resolvable using the methods developed to date and discusses future work.

8.1 COLLABORATIVE CLASSIFICATION NETWORKS

To illustrate possible collaborative relationships between design teams, collaborative classification networks, CCN's, are introduced in this section. The goal is to better understand how classifiers can be used to propagate constraint information between dependent collaborators and to identify the need for new methods as necessary.

All of the previous chapters' applications of classifiers to collaborative design have been demonstrated for just one relationship: one team's simulation results serve as the inputs to another team's simulation. The relationship is shown diagrammatically in Fig. 8.1 where the nodes of the diagram represent the two collaborating teams, and the edge between the nodes has a direction signifying that some or all of the parent node's simulation results serve as input to the child node's simulation. For the UAV demonstration problem, the parent node is the subsystems group and the child node is the systems group.

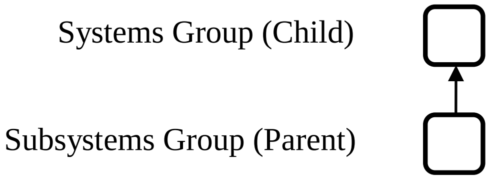


Figure 8.1 CCN for Two Asymmetrically Coupled Design Teams

As more types of relationships are considered, the systems/subsystems distinction becomes blurred and the association with output to input relationships becomes strained. Therefore, relationships designated with a directed edge may also be called asymmetric to emphasize that the simulation that maps design space to parameter space is not easily invertible and to deemphasize any suggestion of a systems/subsystems relationship. The use of a directed edge makes the asymmetric flow of information recognizable in that the parent simulation must be conducted before the child's simulation in order to produce consistent results.

The second type of relationship is the case in which design teams have common inputs to their simulations. For example, the UAV wing design problem was introduced in Chapter 4 as involving two coupled subsystem teams: aerodynamics and structures. Both of these teams required *naca3* and *chord* parameters as inputs to their analyses. This type of relationship is called symmetric, because the shared parameters are design parameters that can be trivially coordinated via prior agreement on their values. The CCN notation for a symmetric relationship between design teams is shown in Fig. 8.2 as a directionless edge.

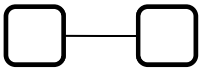


Figure 8.2 CCN for Two Symmetrically Coupled Design Teams

In previous chapters, the aerodynamics and structures design teams were coordinated by effectively merging them into a combined classifier. For a symmetric relationship, the coordination must occur before the coupled teams execute a corresponding simulation. Once the design point has been agreed upon, the two teams can execute their simulations in parallel. In contrast, the asymmetric relationship described in the previous paragraph requires a subsystem simulation to be executed first, followed by the system level simulation. Alternatively, the system level team can generate target values for shared parameters that the subsystem team can try to match.

So far, each node in a CCN has represented a team as depicted by the filleted squares in the previous figures. However, a more precise meaning for the purposes of this research is to use the filleted squares to represent a classifier with a unique set of training points. As illustrated in Fig. 8.3, this allows one to represent the decision to coordinate the training points of separate simulations by merging them into a combined node. For the symmetric relationship between the UAV subsystems the inputs in the *naca3, chord* subsystem domain can be coordinated and the two subsystem teams merged into a single classifier as demonstrated in previous chapters. Likewise, for an asymmetric relationship the systems' *weight, drag* domain can be replaced directly by the subsystems' domain by running the simulations sequentially. Thus, the number of nodes in a CCN represents the number of classifiers that are being independently developed such that their training points do not necessarily coincide with the training points of the other classifiers. The edges represent the ways in which two classifiers are dependent and must be coordinated such that the acceptable class of each classifier captures the

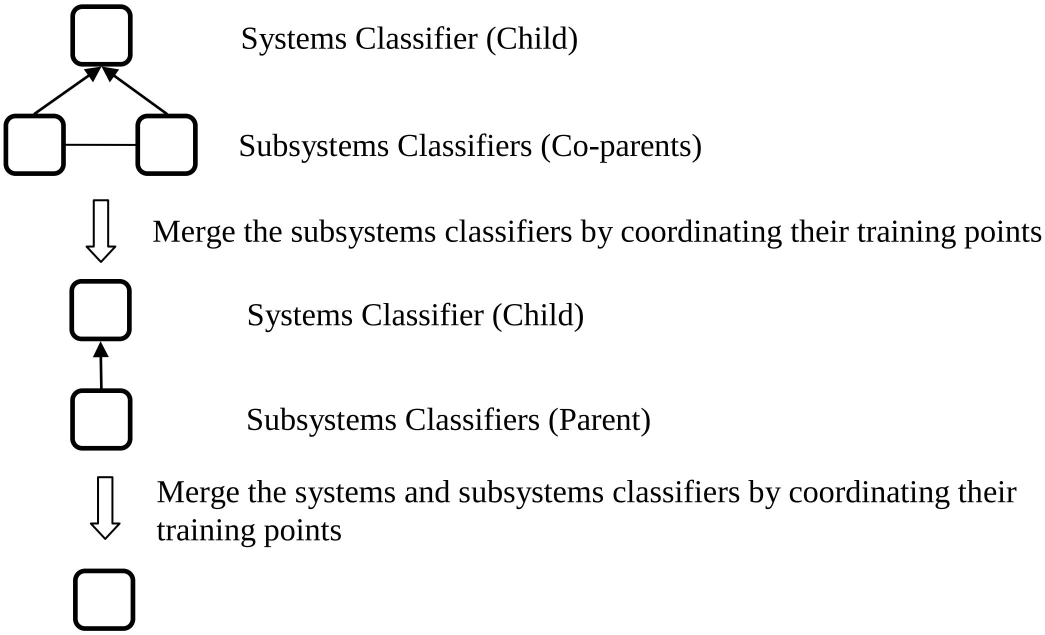

Systems Classifier (Child)
Subsystems Classifiers (Co-parents)
Merge the subsystems classifiers by coordinating their training points
Systems Classifier (Child)
Subsystems Classifiers (Parent)
Merge the systems and subsystems classifiers by coordinating their training points

Figure 8.3 Merging CCN Nodes by Coordinating the Training Points

mutually feasible design space. A CCN is consistent if every classifier has been coordinated such that sampling the mutually feasible design space will result in a design that is acceptable to all system-wide requirements.

The classifier associated with a CCN node can be represented by its Bayesian network as illustrated in Fig. 8.4 for the full UAV distributed design problem. In addition to the class parameter and the design parameters, the performance parameters of the simulation can also be represented. However, because the value of a performance parameter is determined without uncertainty given values for the design parameters, it is represented as a double circle instead of a single circle.

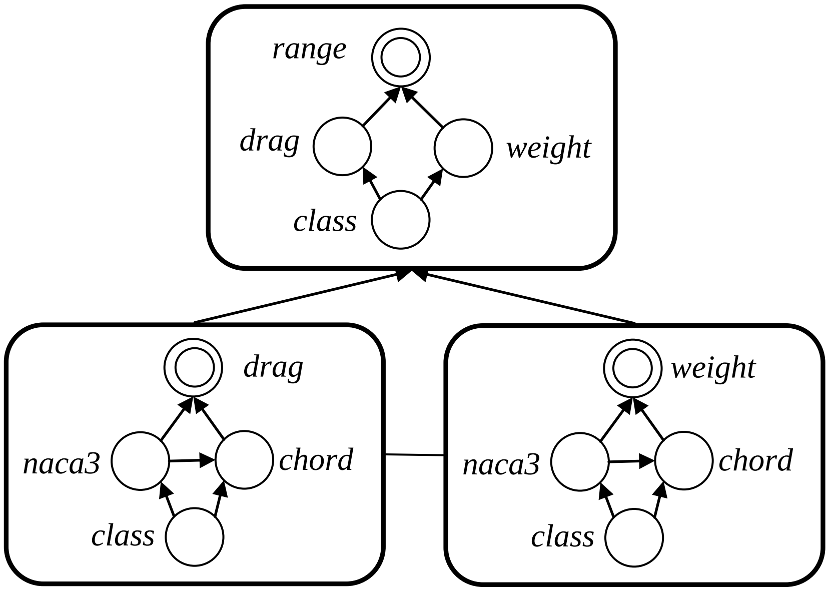


Figure 8.4 CCN with Bayesian Network Classifiers

Just as classifiers can be merged, they can also be split into two or more classifiers with the appropriate edges added to ensure that their dependencies are coordinated. Adding an edge may not always be necessary if it turns out that the classifications can occur independently. For example, two separate classifiers could be created that independently classify values of drag and weight for acceptable values of range if it were known how to limit the values drag and weight take such that all possible combinations of drag and weight will produce an acceptable range. Perhaps an investigation is conducted and upon discovery that the classification decision can be represented using two or more independent classifiers, then the single CCN node can be split into two or more nodes without connecting edges. The common way to do this is to use intervals. As demonstrated in Chapter 4, intervals can lead to irreducible and large classification errors, but if these errors are acceptable, then the CCN connectivity can be simplified.

After defining the basic elements of a CCN as well as a merging/splitting operation, the issue of CCN consistency needs to be addressed. In a consistent CCN, all classifiers have been coordinated such that every classifier's acceptable region is feasible for all classifiers. In order for a CCN to be consistent, constraints on parameters that are local to one classifier must be propagated to the other teams that are joined to it by an edge. The method of propagation of constraints through an asymmetric coupling was demonstrated for the UAV wing design problem in Chapter 4. The method begins with each team sampling their design spaces independently. Each new design point is classified locally as acceptable or not according to all local constraints. In addition, the subsystem team uses the systems' classifier as an additional constraint on the acceptability of their designs. The subsystem design points that classify as acceptable for the systems team and satisfy all subsystem constraints constitute the mutually acceptable set of designs that is then used to train the acceptable design space classifier over the subsystem domain. The subsystem designs within the acceptable design class can then be propagated back up to the systems level by evaluating them with the system level simulation. The propagation flows from the top level requirements down to the subsystem level design definition and then only the best designs as defined by their mutually acceptable classification are sent back up the chain. The process is illustrated in Fig. 8.5. Propagating class through a symmetric relationship is achieved by sharing classifiers and using them as an additional feasibility constraint for each design point. Each classifier's design parameter domain is searched and classified according to local constraints and the constraints of any symmetrically dependent classifier, as shown in Fig. 8.6.

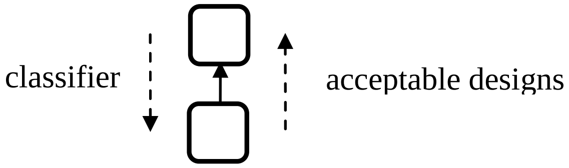


Figure 8.5 Asymmetric Propagation of Class

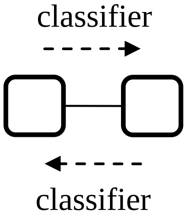


Figure 8.6 Symmetric Propagation of Class

This chapter presents cross-classification as the means to make classifiers that are coupled in the same manner as the full UAV distributed design problem as shown in Fig. 8.4. What is new is that the classifiers at the subsystem level do not have coordinated training points and have not been merged into a single classifier. Furthermore, the two subsystem teams are indirectly coupled through the system level classifier. This occurs in the UAV systems classifier because a different range of weights will be acceptable depending on the value of a design's drag. Resolving the indirect dependency between the subsystems through the systems' classifier requires cross-classification. Resolving the direct dependency between the subsystems through their common design parameters requires symmetric propagation of class. Both cross-classification and symmetric propagation of class are demonstrated in this chapter for resolving the three-way dependency between the UAV teams.

8.2 CROSS-CLASSIFICATION

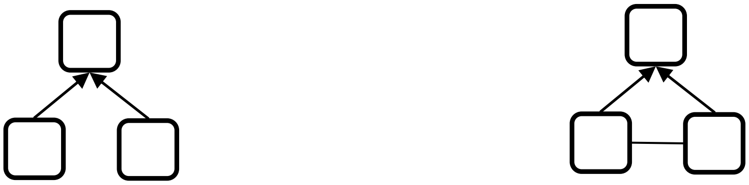


Figure 8.7 Parental Relationships without (left) and with (right) Common Design Parameters at the Subsystem Level

The CCN relationships that are resolved through cross-classification are depicted in Fig. 8.7. There are two cases: the subsystem teams do not (left) or do (right) have common design parameters. As proposed in Section 8.1, the distinction is represented in CCN's as an undirected arc between teams that share simulation inputs. First, the problem of propagating acceptability for the common child, symmetrically uncoupled parent case is considered followed by the common child, symmetrically coupled parent case. But first, a fundamental distinction between the two cases is made: when co-parents share design parameters their design space sampling can be directly coordinated such that they behave as a single unified team. This was the simplification made for all of the previous distributed UAV wing design problems: the design points for the shared *naca3* and *chord* design space were determined simultaneously for both the structures and the aerodynamics teams. When this can be accomplished, the subsystem teams can be merged into a single classifier and the CCN degenerates into a chain that can be resolved by asymmetric propagation of class as covered in Section 8.1 and as demonstrated in previous chapters. The case considered here is when the sampling is not coordinated between the co-parents. Having independent classifiers for each subsystem team allows them to act more independently, choosing which designs to sample and when. In other words, data generated at any time and that has not necessarily been coordinated with co-parents can still be used in the classifier.

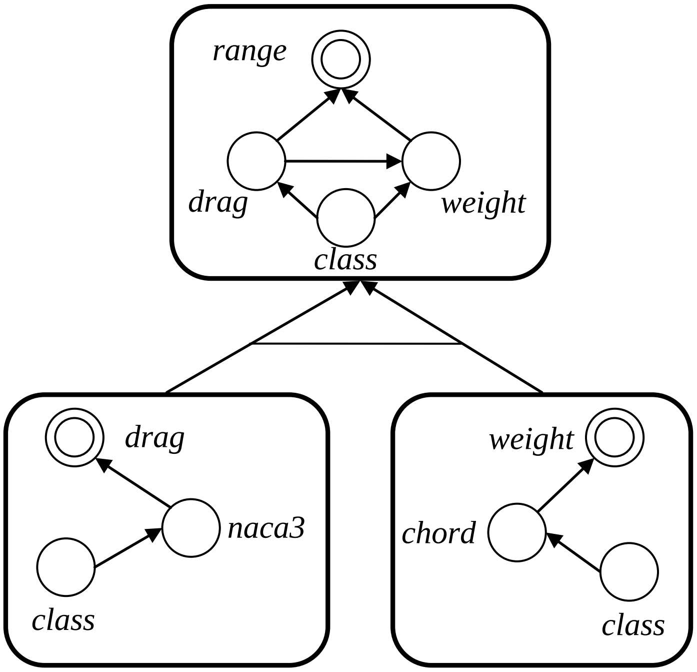


Figure 8.8 CCN without Common Design Inputs at the Subsystems Level

The process for cross-classification is to asymmetrically classify all possible combinations of design points for the different co-parent samples. Consider first the case of symmetrically uncoupled design spaces for the co-parents depicted in Fig. 8.8 as two one-dimensional subsystems spaces. The nomenclature from the UAV design problem is retained for illustrative purposes even though the symmetric independence between the structures and the aerodynamics teams is purely hypothetical. In this hypothetical case, structures has *chord* as its only input and aerodynamics has *naca3* as its only input. An additionaly line connecting the subsystems-to-systems edges in Fig. 8.8 represents dependence through the subsystems teams' common child's classifier and not through the sharing of common domain variables. The structures team chooses *chord* values to evaluate for weight, and aerodynamics chooses values of *naca3* to determine drag using their simulation. In order for structures to classify a *chord* value as being acceptable or not, it needs to be paired with all of aerodynamics' available *naca3* design points and each combination needs to be separately classified as acceptable or not using the systems classifier. Every combination of *naca3* and *chord* that classifies as acceptable to the systems team can be classified as acceptable for both structures and aerodynamics. However, a given value of *naca3* that classifies as acceptable for some combinations with *chord* could also classify as unacceptable with other combinations, making the subsystem classification less clear. The ambiguity can be eliminated by increasing the design space dimensionality to the two dimensional *naca3* by *chord* domain. In the higher dimensional space, all possible combinations are represented by separate points, each of which gets a single classification. The result is a shared classifier between the co-parents that accurately captures all of the cases for which their design parameters combine to acceptable weight and drag system level design points. This case is represented with a connecting line between the subsystem-to-systems edges in the CCN diagram as shown in Fig. 8.8 to indicate the need for cross-classification.

The example used for the case of symmetrically coupled co-parents is the familiar UAV wing design problem with structures and aerodynamics sharing *naca3* and *chord* as their design parameters as shown in Fig. 8.9. In order for the structures team to classify a new design point, (*naca3*, *chord*), they use the systems' classifier to reclassify the aerodynamics design points using their calculation for *weight* resulting from their new design point and combining it with all of the *drag* results from aerodynamics' design points. This use of cross-classification produces a new classifier in terms of *naca3* and *chord* for aerodynamics' training points. Then, the new structures' design point is classified as acceptable or not using the new aerodynamics classifier—an instance of symmetric propagation of class. This procedure approximates the regions of acceptability based upon aerodynamics' choices for *naca3* and *chord* that produce drags that are acceptable when combined with the *weight* calculation from structures and then finds out if structures' (*naca3*, *chord*) design point falls within these acceptable regions.

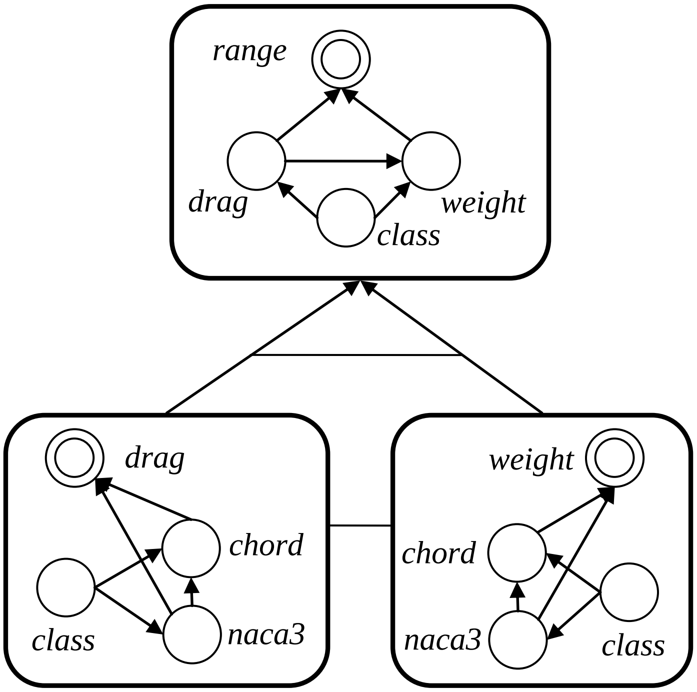


Figure 8.9 CCN with Common Design Inputs for the Co-parents

If this procedure is carried out by structures or aerodynamics, the combinations of *weight* and *drag* that classify as acceptable using the systems classifier will be the same, although the final decision surfaces will be slightly different because of the different sampling sequences in each of their *naca3*, *chord* design spaces. The case of symmetrically coupled co-parents is the same as the previous case of uncoupled co-parents up to and including the point of using the systems' classifier to determine the acceptability of all combinations of subsystem outputs. However, beyond this point, the procedures differ in that symmetrically uncoupled co-parents have to keep track of their parental dependencies through an identical joint classifier of higher dimension whereas symmetrically coupled co-parents will have two separate classifiers of the same dimensionality as before cross-classification—classifiers that require an additional symmetric classification step to establish consistency.

Finally, because the number of points that need to be classified at the systems level is determined by the number of combinations of each of C co-parents' training points the time complexity of cross-classification is $O(N^{C+1}L)$ for the case where all C combined classifiers have N training points, and hence could be prohibitively expensive. As a reminder, L is the longest dependency chain in the BN and hence is greater than zero but less than the number of dimensions. Means for breaking dependencies are therefore important for limiting the difficulty of the problem. Some of the existing methods for breaking dependencies are discussed at the end of the chapter in Section 8.4. But first, the next section demonstrates using cross-classification for resolving the three-way dependency between the teams in the UAV example problem.

8.3 UAV SOLUTION 9: CROSS-CLASSIFICATION AND SEPARATED SUBSYSTEMS

In this section, the UAV wing design problem is solved again with the additional complexity of considering separated subsystem teams each with their own sampling sequence. The solution was implemented in Matlab® and the associated files are included in Appendix G. As with previous solutions, the systems team will again produce an accurate system level classifier before the subsystem teams perform their cross-classification. An additional simplification is made as depicted in the process flow diagram of Fig. 8.12 where the aerodynamics team produces their space-filling sequence before structures begins the cross-classification. With every new sample by structures, all combinations of the resulting weight with the predetermined aerodynamics design points' drag are classified at the systems level for feasibility according to the cross-classification procedure, and then each of structures' (*naca3, chord*) design points is classified again with the new aerodynamics classifier. The resulting classifier and the reported errors are in terms of structures' sequence of design points and their associated classifier. Both the systems and the aerodynamics teams use the Halton sequence for determining their design points, while the structures team uses the purely exploratory adaptive KBN sampling developed in Chapter 7. Using separate sampling sequences for the subsystems teams is central to the desired demonstration of this chapter in order to set it apart from previous solutions. One can also classify the aerodynamics points according to whether or not their drag values are within systems' search domain and then to use this classifier to immediately determine if structures' design point is unacceptable or if cross-classification is needed to make the decision, shown as step 3 in Fig. 8.10.

Example structures' classifiers for both the points within systems' search domain as well as the acceptable points are presented in Fig. 8.11 for the case of $N = 10$, 20 and 100 design point samples. Figure 8.12 presents the total error rates as the sum of the false positive and false negative classification error rates in terms of the 10th, 50th, and 90th percentiles and includes for comparison the baseline error rates from the UAV solution that used the new rule presented at the end of Chapter 5. The total subsystem classification error rates remain under 10% after 100 samples and are comparable to the

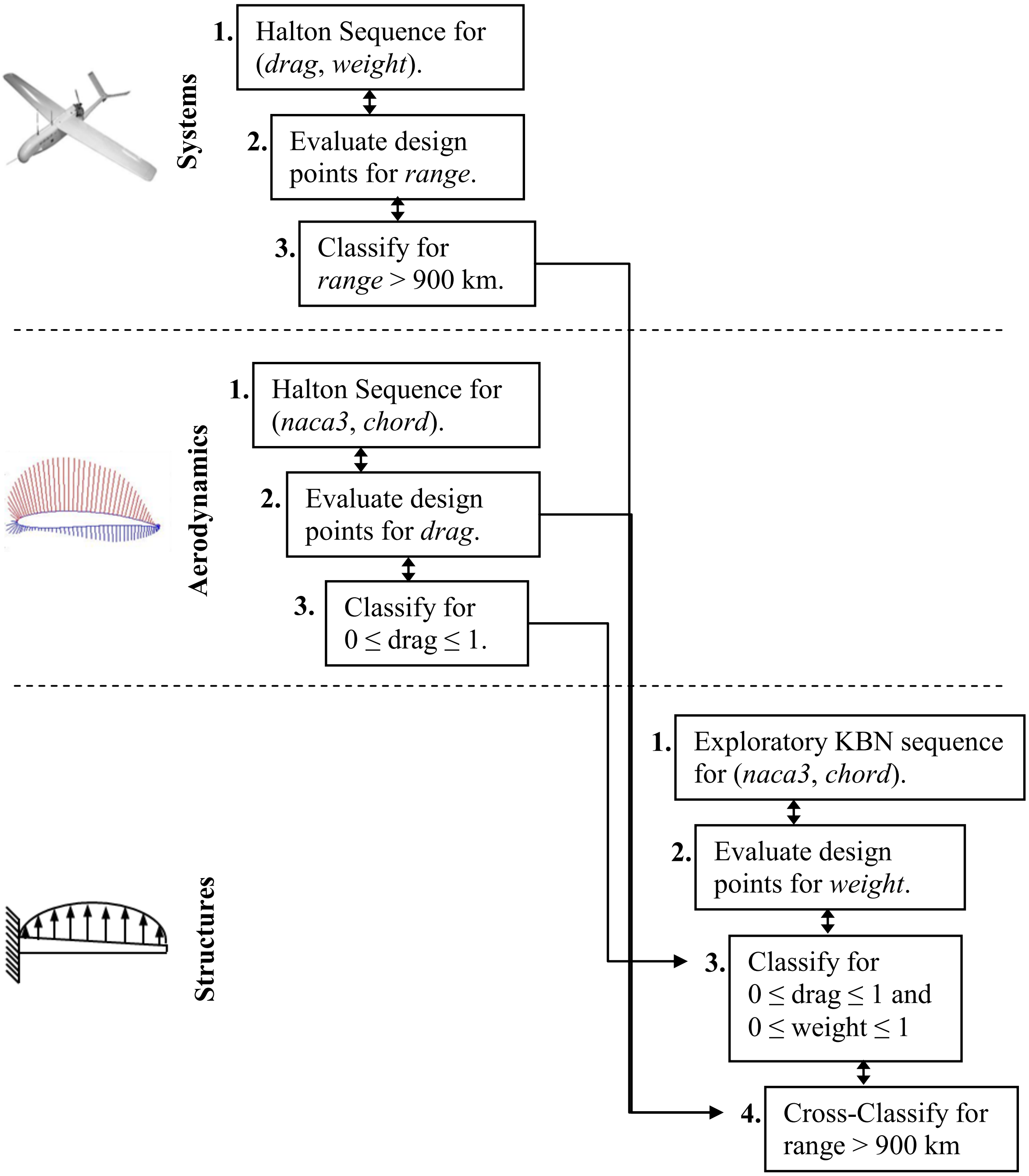

Systems
1. Halton Sequence for (*drag*, *weight*).
2. Evaluate design points for *range*.
3. Classify for *range* > 900 km.
Aerodynamics
1. Halton Sequence for (*naca3*, *chord*).
2. Evaluate design points for *drag*.
3. Classify for 0 ≤ drag ≤ 1.
Structures
1. Exploratory KBN sequence for (*naca3*, *chord*).
2. Evaluate design points for *weight*.
3. Classify for 0 ≤ drag ≤ 1 and 0 ≤ weight ≤ 1
4. Cross-Classify for range > 900 km

Figure 8.10 The UAV Collaborative Design Process

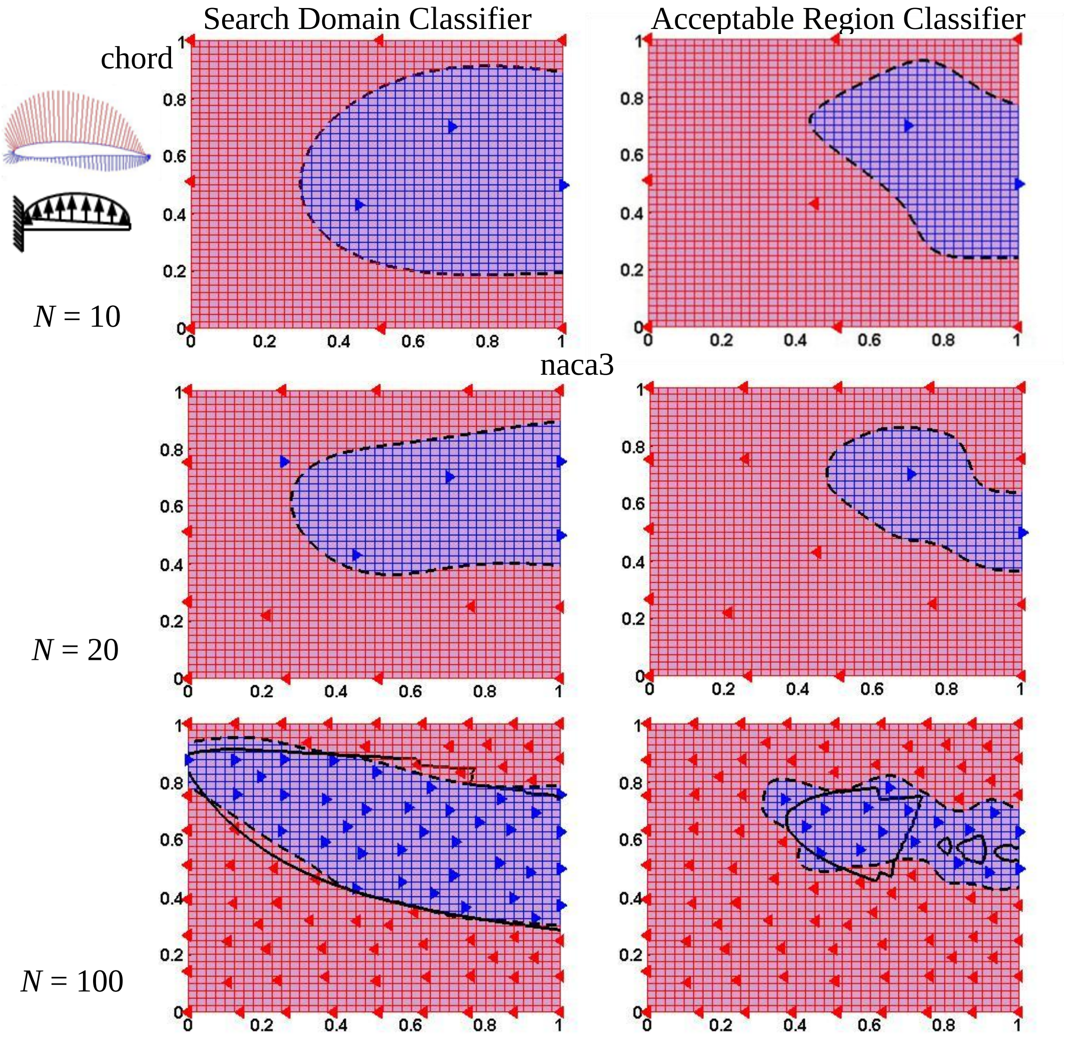


Figure 8.11 Subsystems' Search Domain and Acceptable Region Classifiers

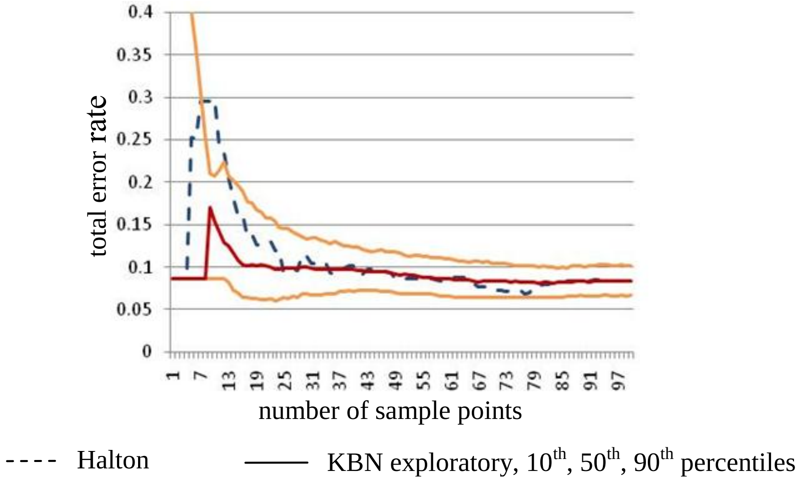


Figure 8.12 KBN Classifier Total Error Rates for the Subsystems

Halton sequence error rates that did not use cross-classification. Figure 8.13 also compares the previous kbn exploratory sampling solution without cross-classification to the solution from this section that used cross-classification. Cross-classification introduced a small amount of additional classification error relative to not using cross-classification.

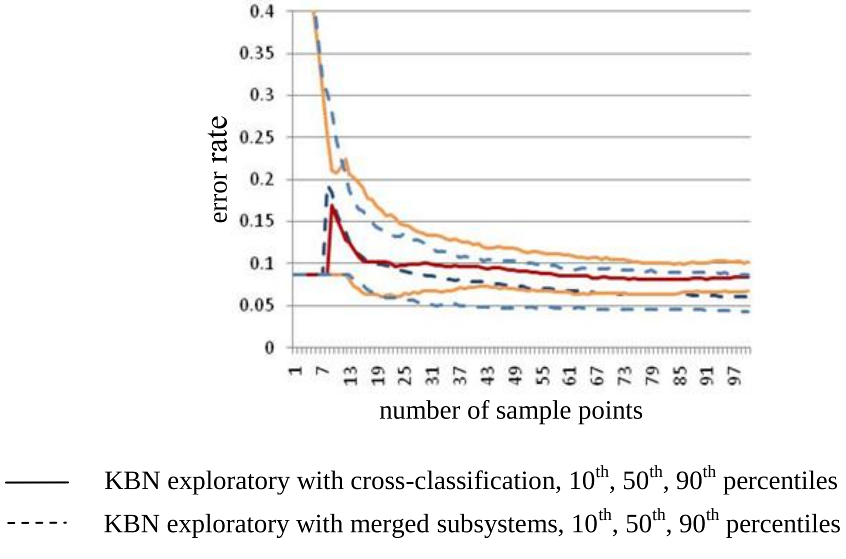
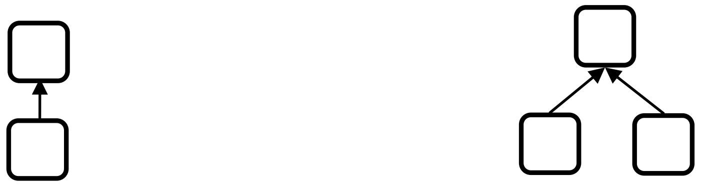


Figure 8.13 KBN Classifier Error Rates for the Subsystems

8.4 DISCUSSION

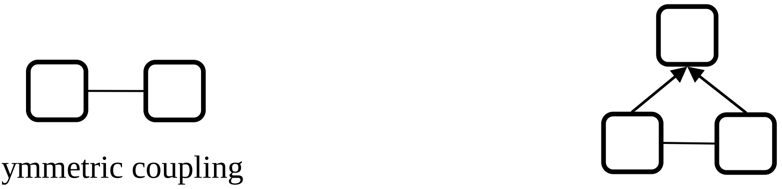
The primary result from this chapter was the development and demonstration of cross-classification for the purposes of propagating classification of acceptability from a systems level requirement that imposes a dependent relationship on the simulation results of two subsystems level teams. For the example UAV wing design problem, cross-classification produced reasonable error rates that are not much different than error rates from processes that did not use cross-classification, using instead the coordination of the subsystem teams' sampling sequences such that they could be considered as one team. The benefit of the new cross-classification arrangement is that the two subsystem teams can independently choose which design points to run through their simulations. Knowledge from simulations runs that were not coordinated can now contribute to the joint classification of both subsystems, providing additional scheduling flexibility. However, the potential cost of cross-classification is high because all combinations of subsystem design points were classified at the systems level. Furthermore, cross-classification dependencies can propagate through a CCN to expand the dimensionality of classifiers as discussed next. There may be ways to help mitigate the cost, including taking advantage of monotonic relationships. However, there are very practical methods already in use that can be used to eliminate the need for cross-classification and the potential dimensional expansion of classifiers altogether by controlling the complexity of the problem. Some of these methods will be discussed later after considering more complex CCN relationships next.

The types of classifier dependencies that can be resolved using the coordination methods demonstrated in this research will be called the admissible CCN's. Inadmissible CCN's are opportunities for future work to define the coordination methods that can make them consistent. Figure 8.14 reviews the CCN dependencies that have been resolved directly through the example problems. On the top left of Fig. 8.14 is the asymmetrically coupled system and subsystem relationship solved in the previous chapters using the asymmetric propagation of class. On the bottom right of Fig. 8.14 is the symmetrically coupled co-parents relationship between the two subsystem teams and their common system level classifier that was solved in the previous section of this chapter using cross-classification. Cross-classification can be separated into two pieces: 1) the classification of every combination of co-parent's training points at the system level and 2) the symmetric propagation of class between the co-parents that share input parameters. These two procedures can hence be separately applied to resolve the two CCN's shown in the top right and the bottom left of Fig. 8.14 respectively.



asymmetric coupling

symmetrically uncoupled co-parents

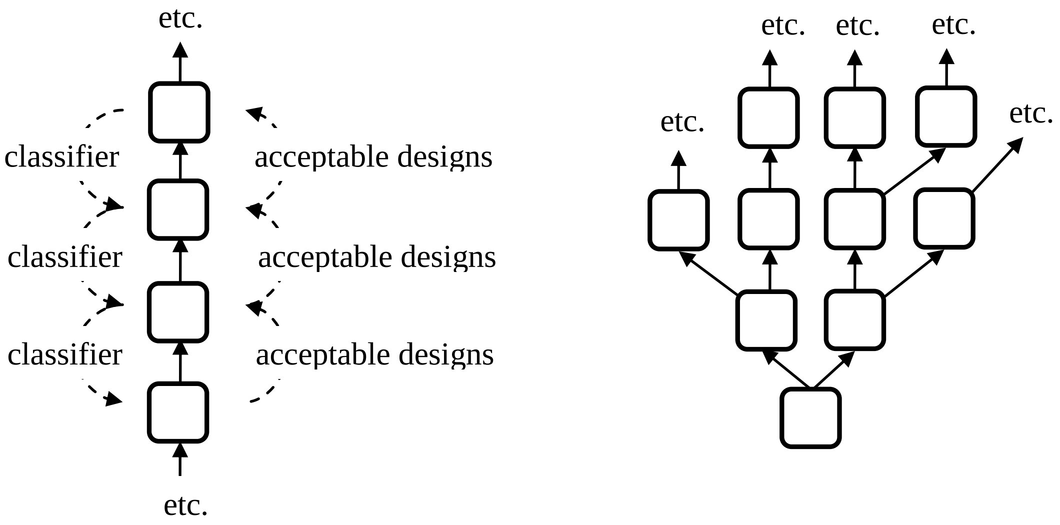


symmetric coupling

symmetrically coupled co-parents

Figure 8.14 Demonstrated Admissible CCN's

Applying the asymmetric propagation of class as demonstrated in the previous chapters easily extends to a chain of asymmetrically coupled classifiers as shown on the left side of Fig. 8.15. Furthermore, every parent can have multiple independent children because each child's classifier will be used in the same manner using the asymmetric propagation of class. This allows for the resolution of a tree structure as shown on the right side of Fig. 8.15. Both of these structures are resolvable using just the asymmetric propagation of class.



chains

trees

Figure 8.15 Admissible CCN Chains and Trees

Of more interest to complex design is the hierarchical structure shown on the right of Fig. 8.16 where systems are made up of multiple subsystems and each subsystem is independent. Hierarchical structures have the useful recursive property of every node having a hierarchical relationship with its ancestors. This property has been taken advantage of by analytical target cascading (ATC) which has a guaranteed convergence for hierarchical dependencies using a recursive coordination strategy as long as the design space is smooth and convex (Michelena et al., 2003). A slightly more general structure called a polytree allows for nodes to have multiple children and parents as long as there are no undirected loops. An undirected loop occurs if more than one path connects any two nodes once the arrows have been removed from the graph's edges to allow for edge traversal in either direction. For this reason, polytrees are also called singly-connected networks (Pearl, 1988). Collaborative design networks (CDN) that use multiply-sectioned Bayesian networks take advantage of polytree structures for finding optimal solutions in an efficient distributed agent-based framework (Xiang et al., 2004). A polytree that is not hierarchical, a tree or a chain is illustrated on the left side of Fig. 8.16. In general these structures are not admissible for the following reasons.

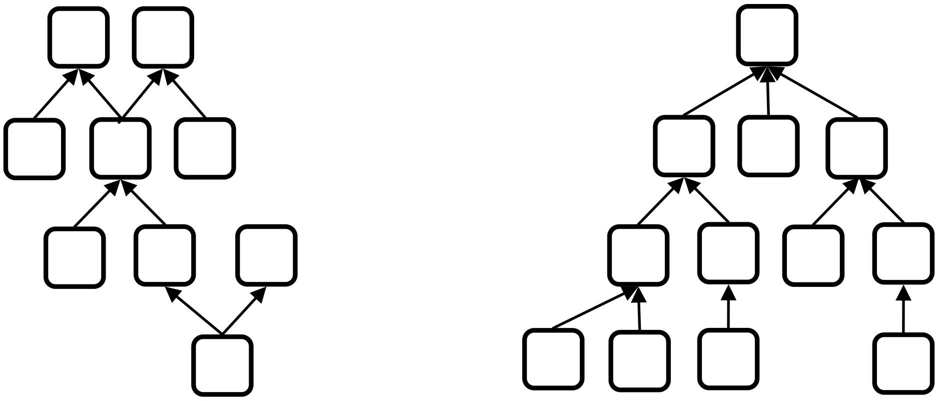


Figure 8.16 CCN Polytrees

Unless the structure is a chain or a tree, making CCN polytrees consistent requires at least one application of cross-classification. As was discussed in Section 8.2, cross-classification between uncoupled co-parents requires merging the co-parent classifiers into a single higher dimensional classifier with training points that are all of the possible combinations of the co-parent training points and which have been classified at their child's level. The trouble with this approach is that the cross-classification dependency will propagate through a polytree. An example of the propagation is shown in Fig. 8.17 using the hierarchical structure shown in Fig. 8.16. At the top of the hierarchy, all three co-parents of the highest node could be dependent through their common child's classifier, as shown with the connecting line between their edges, and merged into a single classifier as shown at the top of Fig. 8.17. This in turn will make all of the nodes at the third row of the hierarchy co-parents that in general will also need cross-classification to resolve their indirect dependencies through their new common child, as depicted by the new connecting lines between their edges. The cycle continues until the whole CCN has degenerated into a chain.

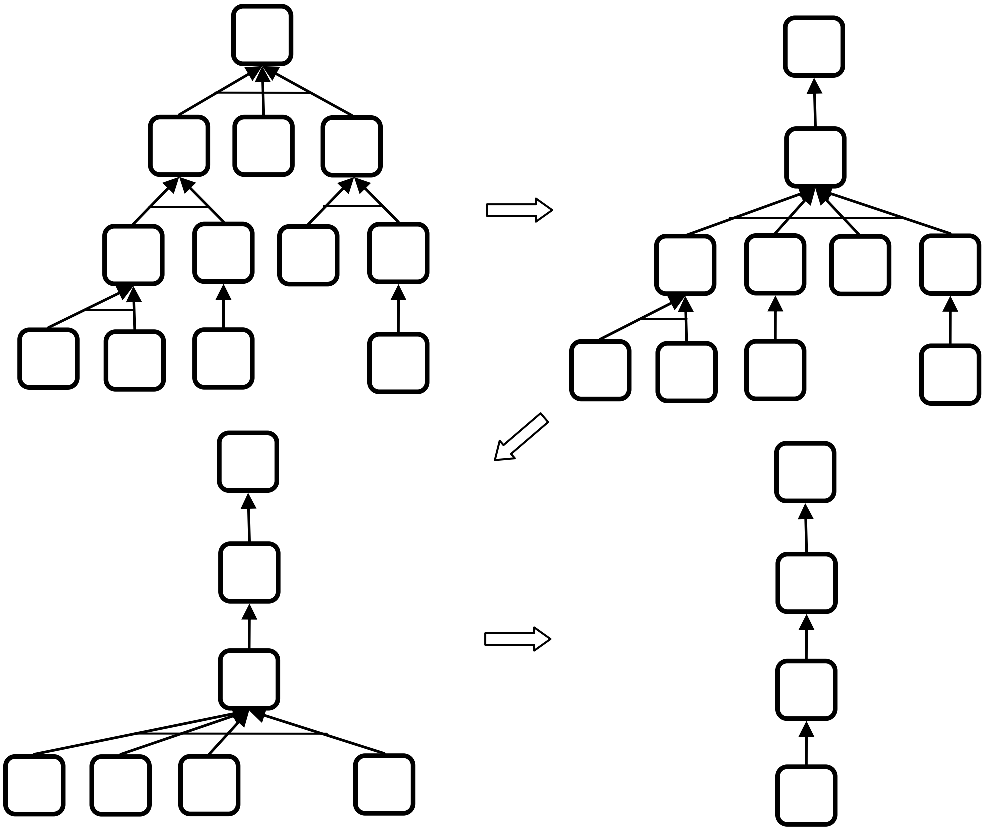


Figure 8.17 Propagation of Dependencies in a Hierarchical CCN

Furthermore, there is no longer necessarily a one to one relationship between the co-parent's outputs and the new merged common child's classifier's domain. For example, when the second row of nodes in the example hierarchy at the upper left of Fig. 8.17 gets merged into a single classifier the middle node will contribute additional dimensionality to the domain of the merged classifier that will not be coupled to any of the outputs of the third row classifiers. This additional dimensionality will need to be brought down to the third row's four dependent classifiers which will hence have the combined dimensionality of five classifiers. The domains of every root node from the original hierarchy will get carried down to the root node of the chain of all of the merged classifiers. For the CCN hierarchy in Fig. 8.18, the chain's root node will have the combined dimensionality of all six root nodes from the original hierarchy shown in the upper left. The complexity increases when the possibility of symmetric coupling is also considered.

While this approach of merging classifiers is possible, the expansion of classifier dimensionality and the associated cost of combinatorially creating the new training points might need to be controlled. Furthermore, some of the independence between groups might want to be preserved for the scheduling benefit of not needing to coordinate the mapping process. Methods are needed to make local classification possible without as complete of a consideration of all of the possible dependencies between the design parameters. This would allow for some of the structure of polytree CCN's to remain and not necessarily collapse into a chain. Some possibilities are discussed next.

There are two ways to limit the propagation of dependencies: with the design and with the design process. The dependencies between groups could be a result of the concept being analyzed. There could be completely new concepts that require new dependencies between new simulations that change the design problem altogether. On the other hand, systems level dependencies are probably general enough to apply to a wide range of concepts such as the *drag* and *weight* dependency for the UAV range. The trade-offs between factors that affect systems level performance will be very common in physical products. While perhaps not always at the systems level, product design can certainly impact the dependencies below the systems level. In particular, product architecture can be modular for the express purposes of reducing design dependencies. Furthermore, part geometries themselves can be changed in order to move load paths or tune stiffness such that design decisions are isolated from each other. These examples are instances of conceptual robustness that seeks to make local design decisions such that dependencies between groups are minimized. Identifying which dependencies are fundamental versus which dependencies are affected by product design and how to use this information is an interesting avenue for future research. Combining results from the very important research on product architecture and modularity with the results from this research would be a very interesting collaboration.

The design process can be conducted such that dependencies between groups are minimized or eliminated. Design decomposition is an important field of work that will obviously greatly impact dependencies between groups and can be used to determine better if not optimal design team boundaries. Hence, another important direction for future research will try to answer the question of how and when to either eliminate an interface or to create a new interface in order to help resolve the problem of propagating class. In addition to decomposition, classification decision boundaries can be chosen such as intervals that eliminate the dependency between groups. A particularly interesting approach that would be a nice complement to this research for determining intervals based upon subsystem achievability is developed in (Liu et al., 2008). However, intervals can lead to classification errors or unnecessarily small acceptable regions of the design space. Hence identifying when intervals are a good choice versus more general classifiers that can capture important dependencies is an important aspect of this research that should be considered in more depth.

Finally, the most common method in practice should be mentioned: the interfaces can be standardized. It is likely that most design processes seek to fix interface parameter values early in the design process in order to proceed without further communication between groups. The express goal of this research is to try to avoid this situation whenever appropriate in order to encourage consideration of a larger set of possible designs. However, there will be interfaces that are not important for the system level performance and can be frozen or standardized without concern for missing potentially better designs. For example the interface between threaded fasteners and the mating parts that they join is a commonly standardized relationship. Recognizing when to freeze an interface and when to explore alternatives is an important aspect of this research that warrants a closer look.

However, there may be new methods for making the classifiers in ploytree and hierarchical dependency structures consistent without removing dependencies. Future work should attempt to identify new procedures for handling these important classes of dependency structures perhaps with the help of a closer study of ATC and CDN's. A judicious use of metamodels might also allow for a significant reduction in the complexity of cross-classification because every combination of subsystems' design points would not need to be classified at the systems level. CCN's as presented in this research are just the start at creating a representation for which the applicability of resolution procedures can be recognized and then implemented. Future work should also more rigorously define the class of admissible CCN structures that a collection of resolution procedures can solve.

Chapter 9. Conclusions

The research presented in the previous chapters is the first step in the direction of a systematic method for realizing a set-based collaborative design process where team dependencies are characterized and resolved more comprehensively than ever before. This research has introduced and validated the possibility of using classifiers towards these ends. One particularly flexible classifier paradigm called kernel-based Bayesian network (KBN) classification was presented and used throughout this research to demonstrate the feasibility of using classifiers for set-based collaborative design. Along the way, the KBN classifier was adapted to further meet the goals of this research's hypothesis in some important ways. The accomplishments toward that end are reviewed next followed by a critical evaluation of these achievements and discussion on future work for advancing the method further. The chapter concludes with a discussion of the practical implications of the research.

9.1 ACHIEVEMENTS

The first step in developing the new method was to apply it to a simple but non-trivial collaborative design problem for a UAV wing. For this problem, it was verified that classification error rates can converge to a reasonably low level within a reasonable number of design points for both teams, including the subsystems team that was using the system level classifier to determine the feasibility of their design choices. The performance of KBN classifiers was compared to intervals that were shown to not converge to low classification error rates because of their fundamentally limited representational capability. This first demonstration problem established the potential benefit of using classifiers over the simpler and more common alternative of using intervals. The capability of using either a Naïve Bayes or a Parzen Window classifier (two common special cases of the more general Bayesian network classifier) for this problem was also demonstrated. In addition, the use of loss factors to trade off false positive error rates for false negative error rates was demonstrated. This ability is an important feature of the proposed classifier for cases when one type of classification error has a more serious implication. It can also be used to tune the size of the mutually feasible region. These achievements were in line with the stated goal of representing and sharing maps of arbitrarily shaped regions of the design space. The KBN classifier's ability to represent disconnected regions of the design space was demonstrated in the outcomes of different experiments throughout the dissertation.

After the first application of the KBN classifier to the demonstration UAV design problem, the importance of setting the standard deviation of the Gaussian kernels was recognized and confronted by developing a new automated approach that is guaranteed not to misclassify the known design points. This achievement advanced the automation of the proposed method in order to more consistently meet designer expectations in terms of correctly classifying the training points. The new approach removes some of the potential burden from the designer who might otherwise have to conduct many experiments in order to find a good setting for the standard deviation. This advance was in line with the stated goal of achieving full automation for the proposed method.

The next achievement of this research was to demonstrate how expert knowledge can be exploited by the classifier to significantly improve classification error rates. This capability was demonstrated in terms of exploiting monotonic relationships in two ways: seeding the training data with points of known classification and using monotonic domination to automatically classify new points without simulating their performance. This advance was in line with the stated goal of being flexible enough to exploit knowledge from external sources, including designers themselves, and validates the importance of this goal.

The next demonstrated achievement was the ability to use the same technology to direct the search through the design space for the purposes of either exploring new regions or exploiting knowledge of the location of good designs to find similar or better designs. The motivation for this achievement comes from the stated goal of exploiting accumulated knowledge of the design space in order to focus resources as quickly as possible. In particular, the ability of the classifier to identify and produce samples from the region in the subsystem design space that maps to the system level search domain was particularly effective in generating design points that are relevant to the systems level team. The novel exploration technique has the additional benefit of adapting to avoid regions that have already been sampled, which further facilitates the method's ability to exploit feasibility information from other sources.

Finally, the use of classifiers for a three-way dependency between design teams called cross-classification was demonstrated. This achievement extended the types of relationships between coupled design teams that can be resolved by sharing classifiers to a total of four configurations as described in Chapter 8. Furthermore, a new graph-based representation was introduced to facilitate a discussion of the relationships between design teams that can and cannot be solved using the methods developed in this research.

9.2 DISCUSSION AND FUTURE WORK

These achievements were significant advances toward realizing the goal of this research to develop a new tool to facilitate set-based collaborative design and they represent a major step forward from the previous work in the literature in several ways. The classifier technology used to map the feasible regions was demonstrated to be more flexible than intervals. This demonstration was important because intervals are simple to implement. Of the existing methods for mapping more complicated feasible regions, the arguments in favor of the proposed method, although not yet demonstrated, are clear. The proposed method does not rely on discretizing the design space and using histograms to represent feasibility. Kernel density methods are more flexible and efficient. Furthermore, classifiers exploit knowledge from all design points and not just the feasible ones. Finally, previous methods, with the exception of design steering, do not integrate the knowledge representation with the search process.

While the proposed method is an advance over the previously proposed methods, future work should evaluate alternatives as well as continue to advance the KBN classifier as proposed throughout this dissertation. Toward this end, more collaborative design problems need to be formulated that cover higher dimensional spaces. Methods for managing the increase in time for both sampling and evaluating classifiers as the number of design points and the dimensionality increases will be important to enable scaling the proposed method to higher dimensional problems. More test problems will also provide evidence for the importance of the first goal of this research: to map arbitrarily shaped and potentially disconnected regions of the design space. It was noted in the third chapter that more flexible technology exists than the proposed KBN classifiers which does not have the universal approximation property of RBF networks. However, it is not clear that the additional overhead of a matrix inversion required to train RBF networks is worth the gain in having a universal approximator. Further studies are required to consider how important it is to have a method that completely achieves the first goal of mapping arbitrarily shaped regions of the design space. Part of this problem must include the diminishing returns of more sample points in terms of smaller error rates and the practical limitations to any attempt to achieve zero error representations.

In summary, while the method developed in this research does not completely meet the first goal of this research, it represents a significant advance over previous methods. Future work can help resolve where the proposed technology should lie on the tradeoff between increased computational expense and reduced classification error. It is clear however that there is no need to take a backward step away from classifiers and return to interval-based approaches including their more representational versions of histograms and 2^k-trees.

Perhaps of higher importance in terms of future work is the extent to which the second and third goals were met. The second goal is to provide a range of automation options from full to no automation. The third goal is to enable designers working in parallel to develop and share their maps as desired. The primary motivation for these goals was to facilitate the success of the tool in an interactive design environment. Full automation displaces the burden from the designer to the computer and provides the service of checking the system-wide feasibility of design choices as soon as possible. No automation means that the method can receive and use any external source of feasibility information in order to make the process potentially more efficient because it is recognized that experts will have specialized knowledge of the problem being solved that should be exploited if possible.

Much of the focus of this dissertation was on making the method more automated such that it could reliably be run in the background without burdening the designer. An important advance toward this end was the method developed to automate setting the standard deviation such that the classifier behaved consistently with its own data. The demonstrations from Chapter 5 suggest that the method has been developed enough to be ready for use although additional work can be done to improve the method as discussed at the end of Chapter 5. The other important advance made by this research toward realizing the full automation goal was the adaptive sampling methods demonstrated in Chapter 7. The methods developed in Chapter 7 have a lot of potential for refinement in future work. In particular, the methods developed need to be tested with higher dimensional problems and adjusted if necessary. Future work could develop guidelines for combinations of explore/exploit strategies that are effective. Future work could also include a classification of not just the feasible region but also the most preferred regions which could subsequently be sampled in the hopes of finding better performing designs. Sampling on or near the decision boundary is also an intriguing possibility for future study.

All of the validating experiments throughout this dissertation were automated and hence the automation has been extensively demonstrated. However, the goal of accommodating no automation was not explicitly demonstrated; nor was the third goal of allowing designers to work in parallel in order to develop and share their classifiers as desired. The method has been developed to the extent necessary that it can now also be used without being automated or with a mixture of interaction and automation for simple problems similar to the UAV wing design problem. The method can accommodate feasibility information from a designer in order to exploit their insights into the problem. The method can also solicit system-wide feasibility from other classifiers to provide feedback to the designer about the feasibility of their choices for some simple relationships. In light of the extensive discussion in Chapter 8 concerning the types of classifier dependencies that can be resolved using the methods developed in this research, it is clear much work needs to be done in order to scale the tools to more complicated systems. More experiments should also be conducted to evaluate how effectively the tool can be used concurrently. However, it should be possible to test the technology as it stands now in an interactive design environment. This is an important next step to completely validate the method's ability to meet the second and third goals of this research. Significant resources will be required to implement the method in a software framework that meets the stated goals of this research. However, the required effort could be mitigated by implementing the method within existing design steering software (Carlson et al., 2008). The importance of this next step is not just to validate the method but also to gain insight into how to improve it further.

Finally, future work should expand the scope of this research that was presented in the first chapter to include more of the design process. The method could be extended to aid designers in evaluating how to spend resources in order to pursue different concepts. Inspiration for this extension can come from (Otto and Wood, 1995; Malak et al., 2009) to name just a couple of precedents. Future work could look into combining variable fidelity information with more inspiration from Q2S2 (Rai and Campbell, 2008). Future work could look into guiding the setting of the softer constraints as well as classifying according to a preference in order to control the size of the feasible region. Future work could look at adapting previously developed classifiers to new but similar problems such that the knowledge can be fully leveraged for re-use. For example, sampling methods that are robust to future changes in requirements or applications could be developed. Future work could help identify opportunities for innovation based upon exploring the impact of violating what are perceived to be hard constraints. Future work could also identify how the product concept and architecture affect the constraint dependencies between designers.

9.3 PRACTICAL IMPLICATIONS

This concluding section returns to the practical motivations of the first chapter of this dissertation in order to relate the results of this research to the concerns of designers in industry. Why might designers want to use the methods developed in this dissertation to tackle a problem they are confronting? This research has the potential to change the relationships between designers and the groups they interact with in two important ways. First, some relationships will be common to all product development processes and these relationships should be thoroughly understood and captured. Second, the choices being made that are new to a design team should benefit from a thorough understanding in terms of how they constrict the design space and how they explore trade-offs.

With the use of tools developed by this research, designers can create thorough classifications of the systems-level design problem in order to understand how subsystem design activities interact to create trade-offs and conflicts with respect to system-level performance. There should be a systems-level view of the problem that is invariant to the different concepts being considered and that can be re-used for many new generations of products. Subsystems-level designers also have relationships with other groups such as manufacturing that might have fairly static requirements that can be mapped and re-used across products. It is quite possible that independently applied rules that come from manufacturing are overly constraining designers, and that independently applied part tolerances are overly constraining the manufacturing and quality control of the parts. Accounting for conditional dependencies should provide greater freedom for both parties. Research activity can be independently conducted and applied to alter the relationship between these teams. Perhaps a more thorough mapping of these relationships can provide direction for future technologies that can improve product performance by redefining the mutually feasible design space.

This research should also help designers that are confronting new design problems for the first time by providing the tools to explore and map the results of simulations and experiments more thoroughly than was previously possible. Designers should be able to gain a more thorough understanding of the mutually feasible design space and how their design choices and constraints shape it. Designers should also aquire an understanding of how dependencies between design activities affect product performance and define trade-offs between potential solutions.

As argued in the opening chapter, these gains in knowledge capture and re-use should translate into shorter product development lead times and potentially higher quality products. Product development times for new projects will be reduced by making sure at least one design is developed that is feasible to all stakeholders, reducing the probability of iteration. Future product development lead times will also be reduced because the re-use of captured knowledge will improve the efficiency of decision making. While these broader claims of the potential impact of this research are still preliminary and unproven, it is hoped that the reader will see enough potential in the methods that have been developed and demonstrated to want to extend this research further toward demonstrating these possibilities.

Appendix A

The Matlab® code for using KBN classifiers is presented in this appendix. Table A.1 lists the functions and includes a brief description of what they are used for. Several utility files are included for running the experiments and generating the plots throughout this dissertation.

Table A.1: Matlab® Functions for using KBN's

kbn( )	The KBN constructor function returns a KBN structure.
kbnAddData( )	Add design points to the KBN structure.
kbnEval( )	Evaluate the probability and its derivative at a design point.
kbnEvalH( )	Calculate the smoothing parameter.
kbnGetErr( )	Calculate the error based upon correct data.
kbnPlot( )	Plot the probability distribution in 1 or 2 dimensions.
createTestPoints( )	Creates a grid of test points for checking the classifier errors.
sampleGrid( )	Creates a grid of points.
dataWrite( )	Writes points to a file.
dataRead( )	Reads points from a file.
halton( )	Generates the Halton sequence with the desired dimensionality and number of points.

```
function n = kbn(Din,Dout,C,varargin)

   n.type = 'kbn'; %a kernel-based Bayesian network

   n.g = cell(Din+Dout,1); %the directed acyclic graph of the BN
   n.gc = 'full'; %the connectivity, defaults to fully connected
   for i=1:Din+Dout
       n.g{i}.p = [];
       n.g{i}.c = [];
   end
```

```
n.Din = Din; %number of inputs
n.Dout = Dout; %number of outputs
n.C = C; %number of classes
n.N = 0; %number of data points
n.Nc = zeros(1,C); %number of data points for each class
n.d = []; %data, N by D
n.w = []; %weights, N by C
n.k = @normal; %kernel function
n.h = []; %smoothing parameter, 1 by C
n.hscale = ones(1,C);
n.hshift = 0;
n.lf = ones(1,C); %loss factors for classification
n.bnd = [zeros(1,Din); ones(1,Din)];

propertyArgIn = varargin;
while length(propertyArgIn) >= 2,
    prop = propertyArgIn{1};
    val = propertyArgIn{2};
    propertyArgIn = propertyArgIn(3:end);
    switch prop
        case 'bnd'
            if size(val,1)==2 && size(val,2)==Din
                n.bnd = val;
            else
                display('Input error for the bounds.');
            end
        case 'k'
            if isa(val,'function_handle')
                n.k = val;
            else
                display('Input error for the kernel function
handle.');
            end
        case 'connect'
            if strcmp(val,'none')
                n.gc = val;
            else
                display('Input error for the connectivity.');
            end
        otherwise
            display('Input error.');
    end
end

n.scale = 1./(n.bnd(2,:)-n.bnd(1,:));
n.shift = n.bnd(1,:);

if strcmp(n.gc,'none')
    %naive bayes
    for i=1:Din+Dout
        n.g{i}.p = [];
```

```
            n.g{i}.c = [];
        end
    elseif strcmp(n.gc,'full')
        %fully connected
        for i=1:Din
            for j=i+1:Din
                n.g{j}.p = [n.g{j}.p i];
                n.g{i}.c = [n.g{i}.c j];
            end
        end
    end
    for i=1:Din
        n.g{i}.c = [n.g{i}.c Din+1:Din+Dout];
    end
    for i=Din+1:Din+Dout
        n.g{i}.p = 1:Din;
        n.g{i}.c = [];
    end

end





function n = kbnAddData(n,d,c)
    M = size(d,1);
    D = n.Din+n.Dout;
    n.d = [n.d; zeros(M,D)];
    n.w = [n.w; zeros(M,n.C)];

    for i=1:M
        n.N = n.N+1;
        n.d(n.N,:) = d(i,:);
        n.w(:,c(i)) = n.w(:,c(i)).*n.Nc(c(i))/(n.Nc(c(i))+1);
        n.Nc(c(i)) = n.Nc(c(i))+1;
        n.w(n.N,c(i)) = 1/n.Nc(c(i));
    end
end





function [ps dpdx dpds] = kbnEval(n,xs,varargin)
    M = size(xs,1); %the number of points to evaluate
    D = n.Din; %the number of dimensions
    if size(xs,2)~=D
        display('The xs do not have n.Din dimensions.');
        return;
    end

    if isempty(varargin)
        C = n.C;    %the number of classes
        c = 1:C;    %the classes
    else
        c = varargin{1};
        C = length(c);
    end

    ps = ones(M,1,C);
    if nargout>1
        dpdx = zeros(M,D,C); %derivatives w.r.t. x
        if nargout>2
            dpds = zeros(M,1,C); %derivatives w.r.t sigma
        end
    end

    for l=1:M
        %store the kernel calculations for reuse
        pKernel = zeros(n.N,D,C);
        %store the conditional probabilities for reuse
        pGivenPa = zeros(1,D,C);
        if nargout>1
            %the derivatives of each of D conditionals
            dpdxGivenPa = zeros(1,D,C);
            if nargout>2
                dpdsGivenPa = zeros(1,D,C);
                radiusSquared = zeros(n.N,n.Din);
            end
        end
        for i=1:D
            %calculate the conditional probabilities, p|pa(p)
            pa = n.g{i}.p; %the array of parents
            paWeight = ones(n.N,1,C);
            paNorm = ones(1,C);
            if nargout>2
                dpdsWeight = zeros(n.N,1,C);
                dpdsSum = zeros(1,C);
            end
            if ~isempty(pa)
                paNorm = zeros(1,C);
                for j=1:n.N
                    for k=1:length(pa)
                        paWeight(j,1,:) =
    paWeight(j,1,:).*pKernel(j,pa(k),:);
                        if nargout>2
                            for ci=1:C
                                dpdsWeight(j,1,ci) =
    dpdsWeight(j,1,ci)+radiusSquared(j,pa(k))/(n.h(c(ci))^3)-
    1/n.h(c(ci));
                            end
                        end
                    end
                    for ci=1:C
```

```
                    paNorm(ci) =
paNorm(ci)+n.w(j,c(ci))*paWeight(j,1,ci);
                    if nargout>2
                        dpdsSum(ci) =
dpdsSum(ci)+n.w(j,c(ci))*paWeight(j,1,ci)*dpdsWeight(j,1,ci);
                    end
                end
            end
        end
        for j=1:n.N
            for ci=1:C
                pKernel(j,i,ci) = n.k(xs(l,i),n.d(j,i),n.h(c(ci)));
                pGivenPa(1,i,ci) = pGivenPa(1,i,ci) +
n.w(j,c(ci)).*paWeight(j,1,ci).*pKernel(j,i,ci);
                if nargout>1
                    dpdxGivenPa(1,i,ci) =
dpdxGivenPa(1,i,ci)+n.w(j,c(ci))*paWeight(j,1,ci)*pKernel(j,i,ci)*(-
(xs(l,i)-n.d(j,i))/(n.h(c(ci))^2));
                    if nargout>2
                        radiusSquared(j,i) = (xs(l,i)-n.d(j,i))^2;
                        dpdsGivenPa(1,i,ci) =
dpdsGivenPa(1,i,ci)+n.w(j,c(ci))*paWeight(j,1,ci)*pKernel(j,i,ci)*(d
pdsWeight(j,1,ci)-
dpdsSum(ci)/paNorm(ci)+radiusSquared(j,i)/(n.h(c(ci))^3)-
1/n.h(c(ci)));
                    end
                end
            end
        end
        %normalize by paNorm
        for ci=1:C
            if paNorm(ci)~=0
                pGivenPa(1,i,ci) = pGivenPa(1,i,ci)/paNorm(ci);
                if nargout>1
                    dpdxGivenPa(1,i,ci) =
dpdxGivenPa(1,i,ci)/paNorm(ci);
                    if nargout>2
                        dpdsGivenPa(1,i,ci) =
dpdsGivenPa(1,i,ci)/paNorm(ci);
                    end
                end
            else
                pGivenPa(ci) = 0;
                if nargout>1
                    dpdxGivenPa(1,i,ci) = 0;
                    if nargout>2
                        dpdsGivenPa(1,i,ci) = 0;
                    end
                end
            end
        end
        %mulitply them together to get p(x)
        ps(l,1,:) = ps(l,1,:).*pGivenPa(1,i,:);
        end

        %sum over the derivatives of each of D conditionals
        if nargout>1
            for k=1:D
                dpdx(l,k,:) =
    dpdx(l,k,:)+dpdxGivenPa(1,k,:).*ps(l,1,:)./pGivenPa(1,k,:);
                if nargout>2
                    dpds(l,1,:) =
    dpds(l,1,:)+dpdsGivenPa(1,k,:).*ps(l,1,:)./pGivenPa(1,k,:);
                end
            end
        end
    end

end


function h = kbnEvalH(n)
    if n.N+n.hshift<=0
        h = ones(1,n.C).*n.hscale;
    else
        h = ones(1,n.C).*n.hscale./((n.N+n.hshift)^(1/n.Din));
    end
end


function [e1 e2] = kbnGetErr(n,xTest,c,cTest)
    %xTest is N by n.Din and is the test design points
    xTest = [n.scale(1)*(xTest(:,1)-n.shift(1)) n.scale(2)*(xTest(:,2)-
    n.shift(2))];

    %cTest is N by 1 and is the correct class index
    N = size(xTest,1);

    %c(1:2) are the two test class indices for testing

    %a false positive results if the n.lf*p(x|c)*P(c) is greater for
    c(1)
    %  but cTest(i,1)==c(2)

    %a false negative results if the n.lf*p(x|c)*P(c) is greater for
    c(2)
    %  but cTest(i,n.Din+1)==c(1)

    %e1 is the percentage of false positives,
    %e2 is the percentage of false negatives

    n1 = 0; %the number of false positives
    n2 = 0; %the number of false negatives
    for i=1:N
        %classify the point
        [cs ps] = kbnEvalC(n,xTest(i,:),c);
        if cs(1)==c(1) && cs(1)~=cTest(i,1)
            n1 = n1+1;
        elseif cs(1)==c(2) && cs(1)~=cTest(i,1)
            n2 = n2+1;
        end
    end

    e1 = n1/N;
    e2 = n2/N;
end


function [x p dp] = kbnPlot(n,res,sf,varargin)
    %this function creates a surface grid for a 1D or 2D kbn
    D = n.Din;
    if length(res)~=D
        display('res must have n.Din+n.Dout elements.');
        return;
    end

    if isempty(varargin)
        C = n.C;    %the number of classes
        c = 1:C;    %the classes
    else
        c = varargin{1};
        C = length(c);
    end

    divs = ceil(1./res);
    step = 1./divs;
    if D==1
        x1 = 0:step(1):1;
        x = x1';
        p = kbnEval(n,x)';
        dp = zeros(1,length(x1));
        if C>1
            dp = sf(1)*p(:,1,c(1))-sf(2)*p(:,1,c(2));
        end
    elseif D==2
        x1 = 0:step(1):1;
        x1Count = divs(1)+1;
        x2 = 0:step(2):1;
        x2Count = divs(2)+1;
        x = [x1' x2'];
        p = zeros(x2Count,x1Count,n.C);
        dp = zeros(x2Count,x1Count);
        for i=1:x1Count
            for j=1:x2Count
```

```
                p(j,i,:) = kbnEval(n,[x1(i) x2(j)]);
                if C==2
                    dp(j,i) = (sf(1)*p(j,i,c(1))-
sf(2)*p(j,i,c(2)))/(sf(1)*p(j,i,c(1))+sf(2)*p(j,i,c(2)));
                end
            end
        end
    else
        display('Only 1 or 2 dimensions can be plotted.');
    end
end


function createTestPoints()

    N = 10000;
    xs = sampleGrid(2,[0 0; 1 5],[99 99],true);
    ys = zeros(N,1);
    for j=1:N
        ys(j,:) = systemsF0(xs(j,:));
    end
    fileName = sprintf('dataTestsystemsN10000.csv');
    fh = fopen(fileName,'w');
    dataWrite(fh,xs,ys);
    fclose(fh);

end


function [xs] = sampleGrid(D,limits,bins,onEdge)

    binsize = (limits(2,:)-limits(1,:))./bins;
    index = zeros(1,D);
    k = 1;

    if onEdge
        while index(D) <= bins(D)
            while index(1) <= bins(1)
                xlow = limits(1,:) + index.*binsize;
                xs(k,:) = xlow;
                index(1) = index(1) + 1;
                k = k+1;
            end
            for j = 1:D-1
                if index(j) >= bins(j)
                    index(j+1) = index(j+1) + 1;
                    index(j) = 0;
                end
            end
        end
    else
```

```
        while index(D) <= bins(D)-1
            while index(1) <= bins(1)-1
                xlow = limits(1,:) + index.*binsize;
                xs(k,:) = xlow+binsize*.5;
                index(1) = index(1) + 1;
                k = k+1;
            end
            for j = 1:D-1
                if index(j) >= bins(j)
                    index(j+1) = index(j+1) + 1;
                    index(j) = 0;
                end
            end
        end
    end

end


function dataWrite(fh,x,y)

    Din = size(x,2);
    Dout = size(y,2);
    N = size(x,1);

    if fh==-1
        display('Invalid file handle.');
        return;
    end

    if Dout>0 && size(y,1)~=N
        display('The number of x and y rows must match.');
    end

    fprintf(fh,'Din,%d\n',Din);
    fprintf(fh,'Dout,%d\n',Dout);
    fprintf(fh,'N,%d\n',N);
    for i=1:N
        for j=1:Din-1
            fprintf(fh,'%g,',x(i,j));
        end
        if Dout>0
            fprintf(fh,'%g,',x(i,Din));
            for j=1:Dout-1
                fprintf(fh,'%g,',y(i,j));
            end
            fprintf(fh,'%g\n',y(i,Dout));
        else
            fprintf(fh,'%g\n',x(i,Din));
        end
    end
```

```
end


function [xs, ys, Din, Dout, N] = dataRead(fh)

    if fh==-1
        display('Invalid file handle.');
        return;
    end

    %read Din
    l = fgetl(fh);
    ls = regexp(l,'\,','split');
    if ~strcmp(ls,'Din')
        display('String ''Din'' not found.');
        return;
    end
    Din = str2double(ls(2));
    if Din<=0 || mod(Din,1)~=0
        display('Din must be a positive integer.');
        return;
    end

    %read Dout
    l = fgetl(fh);
    ls = regexp(l,'\,','split');
    if ~strcmp(ls,'Dout')
        display('String ''Dout'' not found.');
        return;
    end
    Dout = str2double(ls(2));
    if Dout<0 || mod(Dout,1)~=0
        display('Dout must be a non-negative integer.');
        return;
    end

    %read N
    l = fgetl(fh);
    ls = regexp(l,'\,','split');
    if ~strcmp(ls,'N')
        display('String ''N'' not found.');
        return;
    end
    N = str2double(ls(2));
    if N<=0 || mod(N,1)~=0
        display('N must be a positive integer.');
        return;
    end

    %read in the data
    xs = zeros(N,Din);
    ys = zeros(N,Dout);
    for i=1:N
        l = fgetl(fh);
        ls = regexp(l,'\,','split');
        d = str2double(ls);
        xs(i,:) = d(1:Din);
        ys(i,:) = d(Din+1:Din+Dout);
    end

end


function xs = halton(D,N)

    xs = zeros(N,D);
    ps = primes(D+1);
    for j = 1:N
       x = zeros(1,D);
       base = ps;
       index = j*ones(1,D);
       while any(index)
           digit = mod(index,ps);
           x = x + digit./base;
           index = (index-digit)./ps;
           base = base.*ps;
       end
       xs(j,:) = x;
    end

end
```

Appendix B

The Matlab® code for the UAV demonstration problem is presented in this appendix. A large portion of this work is credited to Auhona Hoq and Jonathon Lesage. Table B.1 shows the functions with a brief summary for each. This appendix is divided into three subsections for each of the disciplines: systems, aerodynamics, and structures. Each section has a brief description of the physics used and the numerical approach taken with references to the source material as needed. All of the code is at the end.

Table B.1: Matlab® Functions for the UAV simulations

systemF0( )	Calculates the *range* as a function of wing *weight* and *drag*; calls propulsion( ) and fuselage( ).
propulsion( )	Calculates the *specific fuel burn rate* and the *power* as a function of the *thrust* and the *speed*.
fuselage( )	Calculates the *fuselage weight*, *drag*, and *length* as a function of the *diameter*, *volume* and *speed*.
lineSearchGS()	Golden section line search for minimizing 1D functions.
atmosphere( )	Returns the air *density*, *viscosity*, *pressure* and *temperature* as a function of the *altitude* based upon the US Standard Atmosphere (1976).
aeroF0( )	Calculates the *drag* from the *NACA* 4 digit parameterization, the *chord*, and the *angle of attack*.
aero_profile( )	Generates the panels for use in the calculation of *lift*.
aero_forces( )	Calculates the *lift*, *drag*, *moment*, *panel pressure*, and the *span* given the profile generated by aero_profile( ), the *angle of attack*, the *chord*, the *cruise speed*, and the *altitude*.
aero_flow( )	Calculates the *lift*, *drag* and *moment coefficients* using the profile generated by aero_profile( ), the *angle of attack*, and the *Reynolds number*.
boundary_layer( )	Calculates the *drag coefficient* given an array of the cumulative surface length, the flow velocities, and the *Reynold's number*.
structF0( )	Calculates the *weight* based upon the wing's geometry including a *skin thickness* and the aerodynamics loads.
areaproperties( )	Calculates the *area, moments of inertia,* and the *centroids* of an airfoil.

skin( )	Creates a new profile based upon offsetting a given profile by the *skin thickness*.

Systems:

The systems group is responsible for calculating the *range* of the UAV using the Breguet equation for propeller aircraft, B.1 (Raymer, 2006).

$$range = \left(\frac{speed}{specific\ fuel\ burn\ rate}\right)\left(\frac{lift}{thrust}\right)\log\left(\frac{weight}{weight - fuel\ weight}\right) \tag{B.1}$$

To keep things simple, only the cruise condition was considered with a constant speed of 25 m/s (55.9 mph). During cruise, *lift* equals the *total weight* and *thrust* equals *drag*. A limit was placed on the *total weight* of 35 N (7.9 lbf) and a *payload weight* of 10 N (2.25 lbf) was assumed. The goal is to achieve a range of over 900 km based upon these constraints. As a point of comparison, a UAV called the Spirit of Butts Farm flew over 3000 km across the antlantic and weighed in at under 5 kg (Sohn, 2003). The weight limit will always be active because any remaining weight under the limit can be taken up by more fuel, extending the range. However, more fuel requires a larger fuel tank which increases weight, decreasing the amount of fuel that can be added. Hence an iterative solution is necessary to converge upon the final range. With each iteration, an optimal fuselage is determined as described after the details of the propulsion system are presented next.

The propulsion system was chosen to be a Fox Eagle 74 piston engine because the performance information was readily available from (Lennon, 1996). From this information a cubic polynomial was fit using Excel to get the *fuel burn rate* as a function of the power, Eq. B.3, which in turn is a function of *speed*, *thrust* and *efficiency*, Eq. B.2. 80% efficiency was assumed.

$$power = (speed)(\text{thrust})/(\text{efficiency}) \tag{B.2}$$

$$fuel\ burn\ rate\ \left({}^{cc}/_{min}\right) = -6.9321\frac{power^3}{745.7} - 1.0455\frac{power^2}{745.7} + 40.028\frac{power}{745.7} \tag{B.3}$$

These equations were implemented in the function *propulsion* that has been reproduced later in this appendix. Incedentally, this is a good use of metamodels to decouple design activity between the systems team and the propulsion engineers. The engine weighs 5.284 N, and its drag was assumed to be 1 N. The thrust, being equal to the drag, depends upon a role-up of the total drag according to Eq. B.4.

$$thrust = fuselage\ drag + engine\ drag + 2\ wing\ drag \tag{B.4}$$

The fuselage is optimally designed by the systems group to maximize the *range*. This is achieved by using a golden section (GS) line search algorithm iterating over the *fuselage diameter*. For every GS iteration there is another nested iteration for maximizing how much fuel can be carried while still meeting the weight limit. In the inner loop the *fuel volume* is set to the remaining available weight, Eq. B.5, and a new *fuselage weight* and *drag* are calculated using the *fuselage* function. The *fuselage* function uses a required *fuselage volume*, Eq. B.6, and the *fuselage diameter* of the GS iteration.

$$\begin{aligned} fuel\ weight = {} & weight\ limit - payload\ weight - engine\ weight - \\ & fuselage\ weight - 2wing\ weight \end{aligned} \tag{B.5}$$

$$fuselage\ volume = \frac{payload\ weight}{payload\ density} + \frac{fuel\ weight}{fuel\ density} \tag{B.6}$$

The fuselage is assumed to have a cigar shape as shown in Fig. B.1. Given the fuselage diameter and required volume, the length calculation is straight forward. Assuming a 1mm wall thickness and a material density of 1139 kg/m^3 (Nylon 6), the *fuselage weight* is easily calculated. The drag is calculated according to an empirical equation that was gotten from Hoerner (1965) and that is repeated as Eq. B.7-11. The calculation of the *Reynolds number* depends on calling the *atmosphere* function which implements the standard atmosphere model for *air density* and *viscosity* as a function of *altitude*.

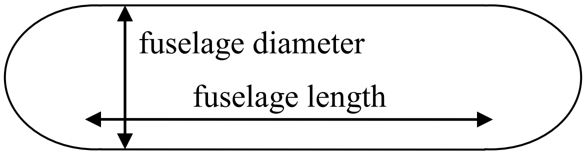


Figure B.1 Fuselage Parameters

$$fineness\ ratio = \frac{fuselage\ diameter}{fuselage\ length + fuselage\ diameter} \tag{B.7}$$

$$Re_l = \frac{(air\ density)(cruise\ velocity)(fuselage\ length + fuselage\ diameter)}{viscocity} \tag{B.8}$$

$$C_{flam} = \frac{1.328}{\sqrt{Re_l}} + \frac{2}{(fineness\ ratio)(Re_l)} \tag{B.9}$$

$$C_D = .33(fineness\ ratio) + C_{flam}\left(\frac{3}{fineness\ ratio} + 3\sqrt{fineness\ ratio}\right) \tag{B.10}$$

$$fuselage\ drag = \frac{1}{2}(air\ density)\left(\frac{\pi(fuselage\ diameter)^2}{4}\right)(cruise\ velocity)^2 \tag{B.11}$$

The total drag calculation will probably underestimate the actual drag because the empennage and the wing to fuselage interface were not accounted for among other things. The remaining details can be found in the Matlab® code that has been provided at the end of this appendix.

Aerodynamics:

The aerodynamics designer is responsible for calculating the wing *lift* and *drag*. The airfoil geometry is parameterized by the 4 digit NACA number (Abbot and von Doenhoff, 1959). The first digit, *naca1*, controls the maximum camber as a percentage of the *chord*. The second digit, *naca2*, controls the distance from the leading edge to the maximum camber in terms of tenths of *chord*. The third digit, *naca3*, controls the maximum thickness of the airfoil in terms of percent *chord*. In addition to these airfoil parameters, the *angle of attack*, *chord* and *span* define the rest of the wing as shown in Fig. B.2. Notice that the *span* is not defined as the more typical wing tip to wing tip distance but instead represents the length of one wing. The planform is rectangular for

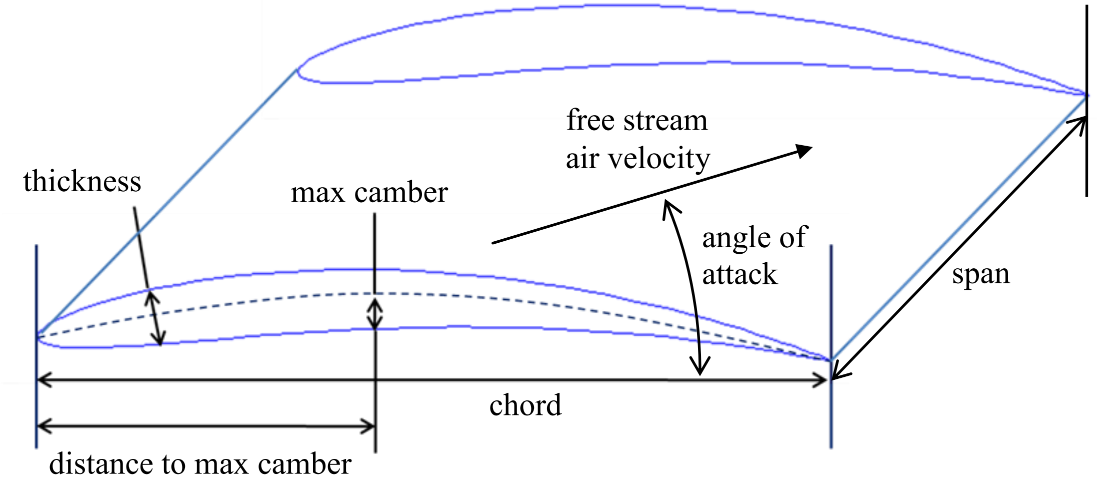


Figure B.2 Wing Parameters

convenience. The *aero_forces* function calculates the wing's lift, drag, moment and span. The *Reynolds number* is calculated according to Eq. B.12 using the *density* and *viscosity* of air as determined by the standard atmosphere model implemented in function *atmosphere*.

$$Re_l = \frac{(air\ density)(cruise\ velocity)(chord)}{viscocity} \tag{B.12}$$

The *lift coefficient* is calculated using the linear vortex panel method as implemented in Katz and Plotkin (2002), Appendix D, Program 7. The *drag coefficient* is calculated using a boundary layer growth model from Moran (1984) implemented in the *boundary_layer* function. Both the *coefficient of lift* and *drag* are modified according to Eq. B.13-14 to account for non-infinite wing spans (Abbott and von Doenhoff, 1959). In Eq. B.13, *m* is the slope of the *lift* versus the *angle of attack* linear relationship. The aspect ratio, *AR*, is equal to the *span* divided by the *chord*. Notice that an Oswald efficiency factor of 1 was used which will make the drag lower than reality.

$$C_{L,total} = \frac{C_L}{1+\frac{m}{\pi AR}} \tag{B.13}$$

$$C_{D,total} = C_D + \frac{{C_L}^2}{\pi AR} \tag{B.14}$$

The span is adjusted to provide enough lift to match half of the total weight, which in this case will always be 17.5 Newtons (3.9 lbf). The wing moment can also be checked and kept under 1 Nm to help with limiting the structural torsion loads as well as placing less demand on any horizontal stabilizers.

Structures:

The structures designer is responsible for calculating the wing weight based upon the wing not yielding under 10X static cruise loads. The most important load is the moment from the lift which is assumed to act halfway down the wing. Each wing carries half of the fixed total 35 Newton weight. Shear stresses are also considered and combined into the Von Mises stress, but they are small. The bending stress is calculated from a general cross-section beam bending formula from Cook and Young (1999) repeated here as Eq. B.15. The coordinate system is set up such that *x* is positive toward the nose of the fuselage, *y* is vertical, and *z* is positive toward the wing tip. Solving for the stress hence requires knowing the moments of inertia.

$$bending\ stress = \frac{M_y I_x - M_x I_{xy}}{I_x I_y - {I_{xy}}^2}(x - x_{centroid}) - \frac{M_x I_y - M_y I_{xy}}{I_x I_y - {I_{xy}}^2}(y - y_{centroid}) \tag{B.15}$$

The only internal geometry that structures can vary is the skin thickness of the wing which is changed until the wing stress under the 10X load is equal to the yield strength. The area properties of the airfoil are found first assuming that the wing is solid, followed by the area properties of the hollow section. The results are combined to get the overall section properties. This occurs in the function *areaproperties* within which the sections are broken up into triangles and combined appropriately. The profile of the outer wing geometry minus the skin thickness is determined in the function *skin*. The details are in the code.

```
function [range] = systemsF0(xIn)

   drag = xIn(1);
   wing_weight = xIn(2);

   %fixed input
```

```
speed = 25; % (m/s)
payload_weight = 10; %N
payload_density = 7063.2; %N/m^3
fuel_density = 7063.2; %N/m^3
engine_weight = 5.28; %N
engine_drag = 1; %N
weight_limit = 35; %N
wingweight_limit = weight_limit-payload_weight-engine_weight;
power_limit = 1350; %W

its = 0;
[range, fuselage_diameter, feasible] =
lineSearchGS(@systems_subfunc,.01,1);
range = -range/1000;
if range<=600
    feasible=0;
end

function [range, fuselage_diameter, sysFeasible] =
systems_subfunc(fuselage_diameter)
    its = its+1;

    %perform the systems design
    sysFeasible = 1;
    fuselage_weight = 0;
    weight = 2*weight_limit;
    payload_volume = payload_weight/payload_density; %m^3
    sysIts = 1;
    while abs(1-weight/weight_limit) > .001 && sysIts < 100
        %fuselage calculations
        fuel_weight = weight_limit - payload_weight - engine_weight
- fuselage_weight - 2*wing_weight;
        fuel_volume = fuel_weight/fuel_density; %m^3
        fuselage_volume = payload_volume + fuel_volume; %m^3
        [fuselage_weight fuselage_drag fuselage_length] =
fuselage(fuselage_diameter, fuselage_volume, speed);
        sysIts = sysIts + 1;
        weight = payload_weight + fuel_weight + engine_weight +
fuselage_weight + 2*wing_weight;
    end
    if sysIts>=100 || fuel_weight <=0
        sysFeasible = 0;
        range = 0;
        return;
    else
        %range calculations
        thrust = fuselage_drag + engine_drag + 2*drag;
        [specific_fuel_burnrate power] = propulsion(thrust, speed);
        weight = payload_weight + fuel_weight + engine_weight +
fuselage_weight + 2*wing_weight;
        lift = weight;
        %Breguet range equation for propeller aircraft
        range =
(speed/specific_fuel_burnrate)*(lift/thrust)*log(weight/(weight-
fuel_weight));
        endurance = range/speed; %seconds
    end
    range = -range;
end

end


function [C power] = propulsion(thrust, speed)

    %Input:
    %thrust should be in Newtons
    %speed should be in m/s

    %Constants:
    %propeller efficiency
    efficiency = 0.8;
    %fuel density in kg/m^3: avgas, see Wikipedia
    fuel_density = 720;
    g = 9.81; %m/s^2

    %Calculations:
    %thrust in Newtons, velocity in m/s, convert from Watts to horse
power
    power = (thrust*speed/745.69987158227)/efficiency; %brake horse
power
    if power>1.5
        fuel_burnrate = 35;
    else
        %fuel burn rate in cubic centimeters per minute (cc/min)
        fuel_burnrate = -6.9321*power^3 - 1.0455*power^2 +
40.028*power;
    end
    %fuel burn rate in N/s
    fuel_burnrate = fuel_burnrate*fuel_density*g/60e6;
    %specific fuel burn rate in s^-1, see Raymer pp. 21-22
    C = fuel_burnrate/thrust;
    %power in Watts
    power = thrust*speed/efficiency;

end


function [weight drag length] = fuselage(diameter, volume, speed)

    %Fixed inputs
    density = 1139.09; %kg/m^3
    thickness = .002; %m
    altitude = 100; %m
    g = 9.81;

    %Fuselage calculations
    area = pi*(diameter/2)^2; %meters^2
    if volume <= (4/3)*pi*(diameter/2)^3;
        length = 0;
    else
        cylinder_volume = volume - (4/3)*pi*(diameter/2)^3;
        length = cylinder_volume/area;
    end

    [airdensity viscocity pressure_inf temperature] =
atmosphere(altitude);
    Re = airdensity*speed*(length+2*diameter)/viscocity;

    fineness_ratio = diameter/(length+2*diameter);
    Cflam = 1.328/sqrt(Re) + 2/(fineness_ratio*Re);
    cd = 0.33*fineness_ratio + Cflam*(3/fineness_ratio +
3*sqrt(fineness_ratio));
    A_fuselage = pi*(diameter/2)^2; %meters^2
    drag = .5*cd*airdensity*A_fuselage*speed^2; %Newtons
    surfacearea = pi*diameter*length+pi*(diameter^2); %meters^2
    weight = surfacearea*thickness*density*g; %Newtons
end


function [fopt, x, feasible] = lineSearchGS(f,xl,xu)
    x0 = xl;
    dx = .01;
    [f0, feas] = f(x0);
    x1 = x0+dx;
    [f1, feas] = f(x1);
    bracketed = 0;
    while ~bracketed
        x00 = x0;
        f00 = f0;
        x0 = x1;
        f0 = f1;
        dx = 1.5*dx;
        x1 = x0+dx;
        if x1>=xu
            break;
        end
        [f1 feas] = f(x1);
        if f1>f0
            bracketed=1;
        end
    end
    if ~bracketed
        x = xu;
        [fopt feasible] = f(x);
        return;
    end
    lb = x00;
    lbf = f00;
    ub = x1;
    ubf = f1;
    gs = (sqrt(5)-1)/2;
    ux = lb+gs*(ub-lb);
    lx = ub-gs*(ub-lb);
    [uf feas] = f(ux);
    [lf feas] = f(lx);

    while (ub-lb)>.0001
        %sprintf('lb = %.4g lbf = %.4g ub = %.4g ubf =
%.4g',lb,lbf,ub,ubf)
        if (uf<=lf)
            lb = lx;
            lbf = lf;
            lx = ux;
            lf = uf;
            ux = lb + gs*(ub-lb);
            [uf feas] = f(ux);
        else
            ub = ux;
            ubf = uf;
            ux = lx;
            uf = lf;
            lx = ub - gs*(ub-lb);
            [lf feas] = f(lx);
        end
    end
    x = (ub+lb)/2;
    [fopt feasible] = f(x);
end


function [density viscocity pressure temperature] =
atmosphere(altitude)

    if (nargin == 0) || (isnan(altitude))
        altitude = 86000;
    elseif altitude < 0
        altitude = 0;
    elseif altitude > 86000
        altitude = 86000;
    end
    altitude = altitude/1000;

    %geopotential altitude conversion
    r0 = 6356766; % radius of the earth at sea level (m)
    H = r0*altitude/(r0+altitude);
    %Table 4: temperature-height profile reference parameters
    %geopotential altitude
    Hb = [0; 11; 20; 32; 47; 51; 71; 84.852]; %(km)
    %temperature-height gradient from i to i+1
    LMb = [-6.5; 0; 1; 2.8; 0; -2.8; -2.0];    %(degK/km)
    %Calculated reference molecular-scale temperatures
    TMb = [288.15; 216.65; 216.65; 228.65; 270.65; 270.65; 214.65;
242.946];  %(degK)
    %Calculated reference pressures
    Pb = [101325 22632 5474.8 868.01 110.9 66.938 3.9564]; %(N/m^2)

    %Calculate the molecular scale temperature
    TM = interp1(Hb,TMb,H,'nearest');
    %Convert to temperature (degK)
    temperature = TM;

    %sea level value of the mean molecular weight (kg/kmol)
    M0 = 28.9644;
    %universal gas constant (N*m/(kmol*degK))
    Rstar = 8314.32;
    %unit geopotential (m^2/(s^2*m'))
    g0prime = 9.80665;

    %Pressure calculation
    index = find(Hb >= H)-1;
    index = index(1);
    if LMb(index) == 0
        pressure = Pb(index)*exp(-g0prime*M0*(H-
Hb(index))/(Rstar*TMb(index)));
    else
        pressure =
Pb(index)*(TMb(index)/TM)^(g0prime*M0/(Rstar*LMb(index)));
    end

    %density kg/m^3
    density = pressure*M0/(Rstar*TM);

    %viscocity kg/m*s
    beta = 1.458e-6;
    S = 110.4;
    viscocity = (beta*temperature^(1.5))/(temperature+S);

end


function [drag] = aeroF0(x)
    %input
    m = 4;       %camber
    p = 5;       %max camber location
    toc = x(1);     %thickness over chord
    chord = x(2);   %meters
    alpha = 0.6;    %angle of attack in degrees
    alpha = alpha*pi/180;  %angle of attack in radians

    %fixed input
    altitude = 100; %meters
    speed = 25;     %m/s
    npanels = 100;
    lift_target = 17.5; %Newtons

    feasible = 1;
    profile = aero_profile(m/100, p/10, toc/100, chord, npanels);
    [lift drag moment pressure span] = aero_forces(profile, alpha,
chord, speed, altitude, lift_target);
    if abs(moment)>1 || abs(lift/lift_target-1)>.001
        feasible = 0;
    end
end


function profile = aero_profile(m, p, toc, chord, npanels)

    %initialize profile datastructure
    profile = zeros(npanels,1);

    %Calculate the airfoil profile
    np = npanels/2;
    for i = 1:np+1
        xoc_oncircle = pi()-(i-1)*pi()/np;
        xoc = 0.5*(1+cos(xoc_oncircle));
        %calculate the camber line and slope
        if xoc < p
            ycoc = (m/(p^2))*(2*p*xoc-xoc^2);
            dycoc = (2*m/(p^2))*(p-xoc);
            theta = atan(dycoc);
        else
            ycoc = (m/((1-p)^2))*(1-2*p+2*p*xoc-xoc^2);
            dycoc = (2*m/((1-p)^2))*(p-xoc);
            theta = atan(dycoc);
        end
        ytoc = toc*(1.4845*sqrt(xoc)-0.63*xoc-1.758*xoc^2+1.4215*xoc^3-
0.5075*xoc^4);
        xu(i) = xoc*chord - ytoc*chord*sin(theta);
        yu(i) = ycoc*chord + ytoc*chord*cos(theta);
        xl(i) = xoc*chord + ytoc*chord*sin(theta);
        yl(i) = ycoc*chord - ytoc*chord*cos(theta);
    end
    profile = [xl(np+1:-1:1)' yl(np+1:-1:1)';
               xu(2:1:np+1)'  yu(2:1:np+1)'];
end
```

```
function [lift drag moment pressures span] = aero_forces(profile,
alpha, chord, speed, altitude, lift_target)

    %flow parameters
    [density viscocity pressure_inf temperature] =
atmosphere(altitude);
    Re = density*speed*chord/viscocity;

    %Calculate the aerodynamic forces
    alphas = [alpha; alpha-3*pi()/180]; %two alphas to find the local
slope of the lift vs. alpha line
    [cp cl cm resultu resultl result_loc cd] =
aero_flow(profile,alphas,Re);
    m0 = (cl(1)-cl(2))/(3*pi()/180);     %the slope of the lift vs.
alpha line

    %calculation of span to get the required lift
    a = cl(1)*pi()*chord;
    b = -2*pi()*lift_target/(density*speed^2);
    c = -2*lift_target*m0*chord/(density*speed^2);
    span = (-b+sqrt(b^2-4*a*c))/(2*a);
    Cl = cl(1)/(1+m0/(pi()*span/chord));
    lift = .5*density*span*chord*Cl*speed^2;

    %correction for induced drag
    cdi = (cl(1)^2)/(pi()*span/chord);
    Cd = cdi + cd(1);
    drag = .5*density*span*chord*Cd*speed^2;
    moment = .5*density*span*chord^2*cm(1)*speed^2;
    pressures = cp(:,1)*(.5*density*speed^2);

end


function [cp cl cm resultu resultl result_loc cd] =
aero_flow(panel,alpha,Re)

    %This algorithm is straight out of "Low Speed Aerodynamics"
    % by Joseph Katz and Allen Plotkin, 2nd ed. Apendix D Program No. 7

    %Number of alphas
    alpha_count = size(alpha,1);

    %Number of panels
    panel_count = size(panel,1)-1;

    %Initialize variables
    center = [];
    angle = [];
    length = [];
    a = [];
    b = [];
    cp = [];
    v = [];
    vn = [];
    xu = [];
    xl = [];
    yu = [];
    yl = [];
    su = [];
    sl = [];

    %Panel center points, normal vector, and length
    for i = 1:panel_count
        center(i,1) = (panel(i+1,1)+panel(i,1))/2;
        center(i,2) = (panel(i+1,2)+panel(i,2))/2;
        angle(i) = atan2(panel(i+1,2)-panel(i,2),panel(i+1,1)-
panel(i,1));
        length(i) = sqrt((panel(i+1,1)-panel(i,1))^2+(panel(i+1,2)-
panel(i,2))^2);
    end

    %Influence Coefficients
    for i = 1:panel_count
        for j = 1:panel_count
            %Convert center point to local panel coordinates
            xt = center(i,1)-panel(j,1);
            yt = center(i,2)-panel(j,2);
            x = xt*cos(angle(j))+yt*sin(angle(j));
            y =-xt*sin(angle(j))+yt*cos(angle(j));

            %Find r1, r2, th1, th2
            r1 = sqrt(x^2+y^2);
            r2 = sqrt((x-length(j))^2+y^2);
            th1 = atan2(y,x);
            th2 = atan2(y,x-length(j));

            %Compute the velocity components
            if i == j
                u1l = -0.5*(x-length(j))/length(j);
                u2l =  0.5*x/length(j);
                w1l = -1/(2*pi());
                w2l = 1/(2*pi());
            else
                u1l = -(y*log(r2/r1)+x*(th2-th1)-length(j)*(th2-
th1))/(2*pi()*length(j));
                u2l = (y*log(r2/r1)+x*(th2-th1))/(2*pi()*length(j));
                w1l = -((length(j)-y*(th2-th1))-
x*log(r1/r2)+length(j)*log(r1/r2))/(2*pi()*length(j));
                w2l = ((length(j)-y*(th2-th1))-
x*log(r1/r2))/(2*pi()*length(j));
            end
            %Transform the velocity components into the global csys
            u1 = u1l*cos(-angle(j))+w1l*sin(-angle(j));
            u2 = u2l*cos(-angle(j))+w2l*sin(-angle(j));
            w1 =-u1l*sin(-angle(j))+w1l*cos(-angle(j));
            w2 =-u2l*sin(-angle(j))+w2l*cos(-angle(j));

            %Compute the coefficients of gamma in the influence matrix
            if j == 1
                a(i,1) = -u1*sin(angle(i))+w1*cos(angle(i));
                tmp_a  = -u2*sin(angle(i))+w2*cos(angle(i));
                b(i,1) =  u1*cos(angle(i))+w1*sin(angle(i));
                tmp_b  =  u2*cos(angle(i))+w2*sin(angle(i));
            elseif j == panel_count
                a(i,j) = -u1*sin(angle(i))+w1*cos(angle(i))+tmp_a;
                a(i,j+1) = -u2*sin(angle(i))+w2*cos(angle(i));
                b(i,j) =  u1*cos(angle(i))+w1*sin(angle(i))+tmp_b;
                b(i,j+1) =  u2*cos(angle(i))+w2*sin(angle(i));
            else
                a(i,j) = -u1*sin(angle(i))+w1*cos(angle(i))+tmp_a;
                tmp_a  = -u2*sin(angle(i))+w2*cos(angle(i));
                b(i,j) =  u1*cos(angle(i))+w1*sin(angle(i))+tmp_b;
                tmp_b  =  u2*cos(angle(i))+w2*sin(angle(i));
            end
        end
        %The boundary conditions
        for k = 1:alpha_count
            rhs(i,k) = cos(alpha(k))*sin(angle(i)) -
sin(alpha(k))*cos(angle(i));
        end
    end

    %The Kutta condition
    a(panel_count+1,1) = 1;
    a(panel_count+1,panel_count+1) = 1;
    rhs(panel_count+1,:) = 0;

    %Solve for vortex strengths
    ainv = inv(a);
    for j = 1:alpha_count
        g(:,j) = ainv*rhs(:,j);
    end

    %Calculate tangential velocities, cp's, cl's, cm's
    cl = zeros(alpha_count,1);
    cm = zeros(alpha_count,1);
    cd = zeros(alpha_count,1);
    cp = zeros(panel_count,alpha_count);
    for k = 1:alpha_count
        clx_sum = 0;
        cly_sum = 0;
        for i = 1:panel_count
            vel = 0;
            for j = 1:panel_count+1
                vel = vel + b(i,j)*g(j,k);
            end
            v(i) =
vel+cos(alpha(k))*cos(angle(i))+sin(alpha(k))*sin(angle(i));
            cp(i,k) = 1-v(i)^2;
            %cl = cl+v(i)*length(i);
            clx = cp(i,k)*length(i)*sin(angle(i));
            cly = -cp(i,k)*length(i)*cos(angle(i));
            clx_sum = clx_sum + clx;
            cly_sum = cly_sum + cly;
            cm(k) = cm(k) + cly*(center(i,1)-.25) - clx*(center(i,2));
        end
        cl(k) = cly_sum*cos(alpha(k))-clx_sum*sin(alpha(k));
        %cd = cly_sum*sin(alpha)+clx_sum*cos(alpha);

        %This algorithm is straight out of
        % 'An Introduction to Theoretical and Computational
Aerodynamics'
        % by Moran.

        %Get the tangential velocities at the nodes
        vn = [];
        vn(2:panel_count) = (v(1:panel_count-1)+v(2:panel_count))./2;
        %The trailing edge gets special attention
        v1 = vn(panel_count)+length(panel_count)*(vn(panel_count)-
vn(panel_count-1))/length(panel_count-1);
        v2 = vn(1)+length(1)*(vn(2)-vn(1))/length(2);
        vn(1) = (v1+v2)/2;
        vn(panel_count+1) = vn(1);

        %Stagnation point  THIS COULD BE OFF FOR WEIRD GEOMETRY
        i = 10;
        while sign(vn(i)) == sign(vn(i+1))
            i = i+1;
        end
        if abs(vn(i+1)) < abs(vn(i))
            i = i+1;
        end
        stag_pnt = i;

        %Separate upper and lower flows
        vu = [];
        vu = vn(stag_pnt:panel_count+1);
        vl = [];
        vl = vn(stag_pnt:-1:1);
        vl = -vl;

        %Get surface points for upper and lower flows
        xu = [];
        xu = panel(stag_pnt:panel_count+1,1)';
        yu = [];
        yu = panel(stag_pnt:panel_count+1,2)';
        xl = [];
        xl = panel(stag_pnt:-1:1,1)';
        yl = [];
        yl = panel(stag_pnt:-1:1,2)';

        %Find surface length
        su = [];
        su(1) = 0;
        for i = 2:panel_count+2-stag_pnt
            su(i) = su(i-1) + length(stag_pnt+(i-2));
        end
        sl = [];
        sl(1) = 0;
        for i = 2:stag_pnt
            sl(i) = sl(i-1) + length(stag_pnt-(i-1));
        end

        %Solve for the boundary layer
        [resultu{k}, itransu, cdu] = boundary_layer(su,vu,Re);
        [resultl{k}, itransl, cdl] = boundary_layer(sl,vl,Re);

        if itransu~=0
            result_loc(k,1:2) = [xu(itransu) yu(itransu)];
        else
            result_loc(k,1:2) = [0 0];
        end
        if itransl~=0
            result_loc(k,3:4) = [xl(itransl) yl(itransl)];
        else
            result_loc(k,3:4) = [0 0];
        end
        cd(k) = cdu + cdl;
    end
end

function [result itrans cd] = boundary_layer(s,ve,Re)

    %Initialize variables
    theta = [];
    vgrad = [];
    h = [];
    cf = [];

    %Get node count
    n = size(s,2);

    %Find velocity gradients
    v1 = ve(3);
    x1 = s(3);
    v2 = ve(1);
```

```
x2 = s(1);
for i = 1:n
    v3 = v1;
    x3 = x1;
    v1 = v2;
    x1 = x2;
    if i < n
        v2 = ve(i+1);
        x2 = s(i+1);
    else
        v2 = ve(n-2);
        x2 = s(n-2);
    end
    fact = (x3-x1)/(x2-x1);
    if i==1
        vgrad(i) = ((v2-v1)*fact+(v3-v1)/fact)/(x3-x2);
    else
        vgrad(i) = ((v2-v1)*fact-(v3-v1)/fact)/(x3-x2);
    end
end

%Laminar flow region
%vgrad(1) = abs(vgrad(1));
theta(1) = sqrt(.075/(Re*vgrad(1)));
i = 1;
laminar_separation = 0;
transition = 0;
itrans = 0;
turbulent_separation = 0;
while 1
    %Thwaites' correlation formulas
    lambda = (theta(i)^2)*vgrad(i)*Re;
    if lambda < -0.0842
        %Laminar separation occurs
        laminar_separation = 1;
        itrans = i;
        h(i) = 2.088 + .0731/(.14+lambda);
        break;
    elseif lambda < 0
        l = .22 + 1.402*lambda + .018*lambda/(.107 + lambda);
        h(i) = 2.088 + .0731/(.14+lambda);
    else
        l = .22 + lambda*(1.57 - 1.8*lambda);
        h(i) = 2.61 - lambda*(3.75 - 5.24*lambda);
    end
    cf(i) = 2*l/(Re*theta(i));
    if i > 1
        cf(i) = cf(i)/ve(i);
    end
    i = i + 1;
    if i > n
        %We've reached the end
```

```
            break;
        end
        if i == 2
            theta(2) = theta(1);
        else
            dth2ve6 = .225*(ve(i)^5 + ve(i-1)^5)*(s(i)-s(i-1))/Re;
            theta(i) = sqrt(((theta(i-1)^2)*ve(i-
1)^6+dth2ve6)/(ve(i)^6));
        end
        %Test for transition
        Rex = Re*s(i)*ve(i);
        Ret = Re*theta(i)*ve(i);
        Retmax = 1.174*(1+22400/Rex)*Rex^.46;
        if Ret >= Retmax
            transition = 1;
            itrans = i;
            lambda = (theta(i)^2)*vgrad(i)*Re;
            if lambda < 0
                h(i) = 2.088 + .0731/(.14+lambda);
            else
                h(i) = 2.61 - lambda*(3.75 - 5.24*lambda);
            end
            break;
        end
    end
    %Build in a transistion condition for h
    if laminar_separation == 1 || transition == 1
        i = itrans;
        if h(i) < 1.3
            h(i) = 1.3;
        elseif h(i) > 1.4
            h(i) = 1.4;
        end
    end

    %Turbulent flow region
    if laminar_separation == 1 || transition == 1
        yy1 = theta(i-1);
        if h(i) > 1.6
            yy2 = 3.3 + 1.5501*(h(i)-.6778)^(-3.064);
        else
            yy2 = 3.3 + 0.8234*(h(i)-1.1)^(-1.287);
        end
        while 1
            dx = s(i)-s(i-1);

            %Runge Kutta
                yt1 = yy1;
                yt2 = yy2;
                h1 = yt2;
                if h1 <= 3.3
                    h(i) = 3;
                elseif h1 < 5.3
                    h(i) = 0.6778 + 1.1536*(h1-3.3)^(-.326);
                else
                    h(i) = 1.1 + 0.86*(h1-3.3)^(-.777);
                end
                rtheta = Re*ve(i-1)*yt1;
                cfturb = .246*(10^(-.678*h(i)))*abs(rtheta)^(-.268);
                yp1 = -(h(i)+2)*yt1*vgrad(i-1)/ve(i-1)+.5*cfturb;
                if h1 <= 3
                    h1 = 3.001;
                end
                yp2 = -h1*(vgrad(i-1)/ve(i-1)+yp1/yt1)+.0306*((h1-3)^(-
.6169))/yt1;
                yt1 = yy1+dx*yp1;
                ys1 = yy1+.5*dx*yp1;
                yt2 = yy2+dx*yp2;
                ys2 = yy2+.5*dx*yp2;
                h1 = yt2;
                if h1 <= 3.3
                    h(i) = 3;
                elseif h1 < 5.3
                    h(i) = 0.6778 + 1.1536*(h1-3.3)^(-.326);
                else
                    h(i) = 1.1 + 0.86*(h1-3.3)^(-.777);
                end
                rtheta = Re*ve(i)*yt1;
                cfturb = .246*(10^(-.678*h(i)))*abs(rtheta)^(-.268);
                yp1 = -(h(i)+2)*yt1*vgrad(i)/ve(i)+.5*cfturb;
                if h1 <= 3
                    h1 = 3.001;
                end
                yp2 = -h1*(vgrad(i)/ve(i)+yp1/yt1)+.0306*((h1-3)^(-
.6169))/yt1;
                yy1 = ys1+.5*dx*yp1;
                yy2 = ys2+.5*dx*yp2;

            theta(i) = yy1;
            if yy2 <= 3.3
                h(i) = 3;
            elseif yy2 < 5.3
                h(i) = 0.6778 + 1.1536*(yy2-3.3)^(-.326);
            else
                h(i) = 1.1 + 0.86*(yy2-3.3)^(-.777);
            end
            rtheta = Re*ve(i)*theta(i);
            cf = .246*(10^(-.678*h(i)))*abs(rtheta)^(-.268);
            if h(i) > 2.4
                turbulent_separation = 1;
                break;
            end
            i = i + 1;
            if i > n
```

```
                break;
            end
        end
    end
    %Return stuff
    if laminar_separation == 1
        result = {sprintf('laminar separation')};
    elseif transition == 1
        result = {sprintf('transition')};
    else
        result = {sprintf('flow is fully laminar')};
    end
    cd = 2*theta(i-1)*ve(i-1)^((h(i-1)+5)/2);
end

function [weight] = structF0(x)
%input
m = 4;       %camber
p = 5;       %max camber location
toc = x(1);     %thickness over chord
chord = x(2);   %meters
alpha = 0.6;   %angle of attack in degrees

%the aero results
[lift, drag, moment, span, feasible] = aero_func([m p toc chord
alpha]);

alpha = alpha*pi/180;  %angle of attack in radians

%fixed input
density = 11174.48; %N/m^3
yieldstrength = 7.93E7; %Pa
loadfactor = 10;

%transformation parameter
pivot = 0.25*chord;

%the aero profile
profile = aero_profile(m/100, p/10, toc/100, chord, 100);

%get the points in the structures format
n = size(profile,1);
x = profile(n:-1:1,1)';
y = profile(n:-1:1,2)';
x_a = (x-pivot).*cos(alpha) + pivot + y.*sin(alpha);
y_a = -(x-pivot).*sin(alpha) + y.*cos(alpha);

%get the max thickness
maxthk = 0;
for i=1:(n+1)/2+1
    if (y(i)-y(n+1-i))>maxthk
```

```
        maxthk = y(i)-y(n+1-i);
    end
end

thickness = .0001;
solid = 0;
safetyfactor = 0;
weight = 0;
feasible = 1;
its = 1;
while abs(safetyfactor-1)>.001 && feasible==1 && its<=100
    %get the area of the outer profile
    [area centroid_x centroid_y inertia_x inertia_y inertia_xy] =
areaproperties(x_a',y_a');
    if area==0
        feasible = 0;
        return;
    end
    %get the inner airfoil points
    if thickness > 0 && solid==0
        [xin yin] = skin(x, y, thickness);
        if size(xin,2)==0
            solid = 1;
        end
        xin_a = (xin-pivot).*cos(alpha) + pivot + yin.*sin(alpha);
        yin_a = -(xin-pivot).*sin(alpha) + yin.*cos(alpha);
    end
    %stress and weight calculation
    if solid==0
        [areain centroidin_x centroidin_y inertiain_x inertiain_y
inertiain_xy] = areaproperties(xin_a',yin_a');
        if area-areain <=0
            solid = 1;
            continue;
        end
        centroidnew_x = (centroid_x*area - centroidin_x*areain)/(area-
areain);
        centroidnew_y = (centroid_y*area - centroidin_y*areain)/(area-
areain);
        inertia_x = (inertia_x + (centroid_y^2)*area) - (inertiain_x +
(centroidin_y^2)*areain) - (centroidnew_y^2)*(area-areain);
        inertia_y = (inertia_y + (centroid_x^2)*area) - (inertiain_y +
(centroidin_x^2)*areain) - (centroidnew_x^2)*(area-areain);
        inertia_xy = (inertia_xy + (centroid_x*centroid_y)*area) -
(inertiain_xy + (centroidin_x*centroid_y)*areain) -
(centroidnew_x*centroidnew_y)*(area-areain);
        centroid_x = centroidnew_x;
        centroid_y = centroidnew_y;
        area = area - areain;
    end
    weight = area*span*density;
    moment_x = loadfactor*lift*span/2;
    moment_y = drag*span/2;
    stress = [];
    for i=1:size(x_a,2)
        stress(i) = ((moment_y*inertia_x-
moment_x*inertia_xy)/(inertia_x*inertia_y-inertia_xy^2))*(x_a(i)-
centroid_x) - ...
                    ((moment_x*inertia_y-
moment_y*inertia_xy)/(inertia_x*inertia_y-inertia_xy^2))*(y_a(i)-
centroid_y);
    end
    stress_zz = max(stress);
    stress_yz = loadfactor*lift/area;
    stress_zx = drag/area;
    wingstress = sqrt(stress_zz^2 + 6*(stress_yz^2 +
stress_zx^2))/(sqrt(2));
    safetyfactor = yieldstrength/wingstress;
    if solid==1 && safetyfactor<1
        feasible = 0;
    else
        thickness = thickness/safetyfactor;
    end
    its = its + 1;
end
if its>100
    feasible = 0;
end

end


function [area centroid_x centroid_y inertia_x inertia_y
inertia_xy]=areaproperties (x,y)
    n = size(x,1);
    area=0;
    centroid_x = 0;
    centroid_y = 0;
    inertia_x = 0;
    inertia_y = 0;
    inertia_xy = 0;
    if mod(n,2)
        limit = ((n-1)/2)-1;
    else
        limit = (n-2)/2;
    end
    for i=1:limit
        area1 = .5*abs(det([x(i) x(i+1) x(n-i+1);y(i) y(i+1) y(n-i+1);1
1 1]));
        area2 = .5*abs(det([x(i+1) x(n-i) x(n-i+1);y(i+1) y(n-i) y(n-
i+1);1 1 1]));
        area = area + area1 + area2;

        centroid1_x = (x(i)+x(i+1)+x(n-i+1))/3;
        centroid2_x = (x(i+1)+x(n-i)+x(n-i+1))/3;
        centroid_x = centroid_x + centroid1_x*area1 +
centroid2_x*area2;
        centroid1_y = (y(i)+y(i+1)+y(n-i+1))/3;
        centroid2_y = (y(i+1)+y(n-i)+y(n-i+1))/3;
        centroid_y = centroid_y + centroid1_y*area1 +
centroid2_y*area2;

        inertia_x = inertia_x + ((y(i)^2+y(i+1)^2+y(n-i+1)^2)/12 +
(centroid1_y^2)*3/4)*area1;
        inertia_x = inertia_x + ((y(i+1)^2+y(n-i)^2+y(n-i+1)^2)/12 +
(centroid2_y^2)*3/4)*area2;
        inertia_y = inertia_y + ((x(i)^2+x(i+1)^2+x(n-i+1)^2)/12 +
(centroid1_x^2)*3/4)*area1;
        inertia_y = inertia_y + ((x(i+1)^2+x(n-i)^2+x(n-i+1)^2)/12 +
(centroid2_x^2)*3/4)*area2;
        inertia_xy = inertia_xy + ((x(i)*y(i)+x(i+1)*y(i+1)+x(n-
i+1)*y(n-i+1))/12 + centroid1_x*centroid1_y*3/4)*area1;
        inertia_xy = inertia_xy + ((x(i+1)*y(i+1)+x(n-i)*y(n-i)+x(n-
i+1)*y(n-i+1))/12 + centroid2_x*centroid2_y*3/4)*area2;
    end
    if area~=0
        centroid_x = centroid_x/area;
        centroid_y = centroid_y/area;
    end
    %use the parallel axis theorem to find the inertia about the
centroid
    inertia_x = inertia_x - area*centroid_y^2;
    inertia_y = inertia_y - area*centroid_x^2;
    inertia_xy = inertia_xy - area*centroid_x*centroid_y;
end

function [xin yin] = skin(x, y, thickness)
    change_x=0;
    n = size(x,2);
    for i=1:n-1
        a(i)=atan2((y(i+1)-y(i)),(x(i+1)-x(i)));
    end
    gamma(1)=(pi/2)+a(1);
    for i=2:n-1
        if sign(a(i))==-1 && sign(a(i-1))==1 && a(i-1)>pi/2
            gamma(i)=-(pi/2)+((a(i-1)+a(i))/2);
        else
            gamma(i)=(pi/2)+((a(i-1)+a(i))/2);
        end
    end
    gamma(n)=(pi/2)+a(n-1);
    change_x=cos(gamma);
    change_y=sin(gamma);
    xin=x+(thickness*change_x);
    yin=y+(thickness*change_y);
    while yin(1)<= yin(n)
```

```
        if n<4
            thickness = 0;
            xin = [];
            yin = [];
            break;
        end
        xin(n)=[];
        yin(n)=[];
        xin(1)=[];
        yin(1)=[];
        n = n-2;
    end
    intersect = 0;
    if thickness > 0
        for i=1:(n-1)/2
            if yin(i)<yin(n+1-i)
                intersect = i;
                break;
            end
        end
        if intersect ~=0
            yin = [yin(1:intersect-1) yin(n+2-intersect:n)];
            xin = [xin(1:intersect-1) xin(n+2-intersect:n)];
        end
    end
end
```

Appendix C

The Matlab® code for Chapter 4's UAV solutions 1 and 2 is presented in this appendix.

Table C.1: Matlab® Functions for UAV Solutions 1 and 2

systemsKBN01( )	The first baseline solution for the systems group.
subsystemsKBN01( )	The first baseline solution for the subsystems group.
systemsKBN02( )	Fully disconnected (Naïve Bayes) KBNC.
subsystemsKBN02( )	Fully disconnected (Naïve Bayes) KBNC.
systemsInt0( )	Uses interval classifiers for the systems group.
subsystemsInt0( )	Uses interval classifiers for the subsystems group.
systemsKBN03( )	With loss factors.
demo_sys01( )	Plots runs of systemsKBN01, e.g. Fig. 4.4, 4.5 left.
demo_subsys01( )	Plots runs of subsystemsKBN01, e.g. Fig. 4.8, 4.9 left.
demo_sys02( )	Plots runs of systemsKBN02, e.g. Fig. 4.5 middle.
demo_subsys02( )	Plots runs of subsystemsKBN02, e.g. Fig. 4.8, 4.9 middle.
demo_sys_Intervals( )	Plots runs of systemsInt0, e.g. Fig. 4.5 right.
demo_subsys_Intervals( )	Plots runs of subsystemsInt0, e.g. Fig. 4.9 right.
demo_sys03( )	Plots runs of systemsKBN03, e.g. Fig. 4.12 right.
experiment01( )	Generates errors for systemsKBN01 and subsystemKBN01, e.g. Fig. 4.6, 4.10.
experiment02( )	Generates errors for systemsKBN02 and subsystemKBN02, e.g. Fig. 4.6, 4.10.
experimentInt0( )	Generates errors for systemsKBN01 and subsystemKBN01, e.g. Fig. 4.6, 4.10.
experiment03( )	Generates errors for systemsKBN03 and subsystemKBN01, e.g. Fig. 4.13.

Unless stated otherwise, all solutions used a fully connected KBNC without loss factors. The standard deviations are rule-based using Eq. 3.16 with the scaling factors set according to Table 4.2. The sampling is a Halton sequence.

```
function n = systemsKBN01(n,M)

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based.

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = .25*ones(1,n.C);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/n.Din));
        end
    end

end


function [n] = subsystemsKBN01(n,M,nsys)
```

```
    classifyType = 1;
    % 1 for using the systems classifier
    % otherwise use the systems simulation

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based.

    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        n.hscale = [.1 .5];
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        if classifyType
            %classify the point using systems' classifier
            [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
            if ci==0
                ci = 2;
            end
        else
            %or classify correctly using the systems' simulation
            range = systemsF0(yi);
            if range>=900
                ci = 1;
            else
                ci = 2;
            end
        end
        n = kbnAddData(n,[xi yi],ci);

    end

    %set the standard deviation
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/n.Din));
        end
    end

end
```

```
function n = systemsKBN02(n,M)

    %The classifier is naive Bayes
    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based.

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5],'connect','none');
        n.hscale = .5*ones(1,n.C);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/n.Din));
        end
    end

end


function [n] = subsystemsKBN02(n,M,nsys)

    classifyType = 1;
    % 1 for using the systems classifier
    % otherwise use the systems simulation

    %The classifier is naive Bayes
    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based.
    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4],'connect','none');
        n.hscale = [.1 .5];
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        if classifyType
            %classify the point using systems' classifier
            [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
            if ci==0
                ci = 2;
            end
        else
            %or classify correctly using the systems' simulation
            range = systemsF0(yi);
            if range>=900
                ci = 1;
            else
                ci = 2;
            end
        end
        n = kbnAddData(n,[xi yi],ci);

    end

    %set the standard deviation
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/n.Din));
        end
    end

end


function int = systemsInt0(int,M)

    %Intervals for these values are approximated based upon sampling.
    %The interval is constructed in a balance between errors by using a
max weight
```

```
    %and drag that are weighted averages of the bounds.

    %The search strategy is exploratory only using a halton sequence.

    rangeDB = 900;
    if isempty(int)
        int = createInt();
    end

    for i=1:M

        %get the next data point
        xhalton = halton(int.Din,int.N+1);
        xi = xhalton(int.N+1,:);
        yi = systemsF0(xi./int.scale+int.shift);
        int.d(int.N+1,:) = [xi yi];
        int.N = int.N+1;

        %set the new decision boundary
        if yi>=rangeDB
            if xi(1)>int.dbUB(1)
                int.dbUB(1) = xi(1);
            end
            if xi(2)>int.dbUB(2)
                int.dbUB(2) = xi(2);
            end
        else
            if xi(1)<int.dbLB(1)
                int.dbLB(1) = xi(1);
            end
            if xi(2)<int.dbLB(2)
                int.dbLB(2) = xi(2);
            end
        end
        int.db = int.pct.*int.dbUB+(1-int.pct).*int.dbLB;
    end

    function intOut = createInt()
        intOut.Din = 2;
        intOut.Dout = 1;
        intOut.pct = [.75 .75]; %where to set the decision boundary
        %setting it to [0 0] is a worse case of the minimum
unacceptable weights and drags
        %setting it to [1 1] is a best case of the maximum acceptable
weights and drags
        intOut.bnd = [0 0; 1 5];
        intOut.scale = 1./(intOut.bnd(2,:)-intOut.bnd(1,:));
        intOut.shift = intOut.bnd(1,:);
        intOut.dbUB = zeros(1,intOut.Din);
        intOut.dbLB = ones(1,intOut.Din);
        intOut.db = intOut.pct;
        intOut.d = [];
        intOut.N = 0;
    end

end

function int = subsystemsInt0(int,M,intsys)

    %Intervals for these values are approximated based upon sampling.
    %The interval is constructed as the minimum circumscribing
rectangle

    %The search strategy is exploratory only using a halton sequence.

    if isempty(int)
        int = createInt();
    end

    for i=1:M

        %get the next data point
        xhalton = halton(int.Din,int.N+1);
        xi = xhalton(int.N+1,:);
        yi(1) = aeroF0(xi./int.scale+int.shift);
        yi(2) = structF0(xi./int.scale+int.shift);
        int.d(int.N+1,:) = [xi yi];
        int.N = int.N+1;

        %set the new decision boundary
        %minimum circumscribed rectangle
        yi = (yi-intsys.shift).*intsys.scale;
        if yi(2)<=intsys.db(2) && yi(1)<=intsys.db(1)
            %is it incorrectly classified?
            %then correct decision boundary
            if xi(1)<int.dbLB(1) %left
                int.dbLB(1) = xi(1);
            end
            if xi(1)>int.dbUB(1) %right
                int.dbUB(1) = xi(1);
            end
            if xi(2)<int.dbLB(2) %lower
                int.dbLB(2) = xi(2);
            end
            if xi(2)>int.dbUB(2) %upper
                int.dbUB(2) = xi(2);
            end
        end

    end

    function intOut = createInt()
```

```
        intOut.Din = 2;
        intOut.Dout = 2;
        intOut.bnd = [1 .1; 15 .4];
        intOut.scale = 1./(intOut.bnd(2,:)-intOut.bnd(1,:));
        intOut.shift = intOut.bnd(1,:);
        intOut.dbUB = zeros(1,intOut.Din);
        intOut.dbLB = ones(1,intOut.Din);
        intOut.d = [];
        intOut.N = 0;
    end

end


function n = systemsKBN03(n,M)

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based with loss factors

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = .25*ones(1,n.C);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %update the loss factors
        n.lf(2) = 1000/(n.N^(1/2));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/n.Din));
        end
    end

end


function demo_sys01()

    %halton sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys] = dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = systemsKBN01(n,M);

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1, ph1, dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_subsys01()

    %halton sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations
    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN01(struct([]),100);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = subsystemsKBN01(n,M,nsys);

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

end


function demo_sys02()

    %halton sequence at the systems level
    %rule-based sigmas
    %fully disconnected BN

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys,DinTestSys,DoutTestSys,NtotTestSys] =
dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = systemsKBN02(n,M);
        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

%surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

%surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_subsys02()

    %halton sequence at the systems level
    %rule-based sigmas
    %fully disconnected BN

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN02(struct([]),100);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = subsystemsKBN02(n,M,nsys);

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1, ~, dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

%surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

%surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

end
```

```
function demo_sys_Intervals()

    %halton sequence at the systems level
    %rule-based intervals

    I = 100; %repeat I times
    M = 1; %the number of data points per each of I iterations

    figure();
    ah = axes();
    rh = -1;

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys] = dataRead(fh);
    fclose(fh);

    int = struct([]); %start with the "empty" interval structure
    rangeDB = 900;
    for i=1:I
        int = systemsInt0(int,M);

        cla(ah);
        axis([0 1 0 1]);
        %plot the data points
        for j=1:int.N
            if int.d(j,3)>=rangeDB

plot(int.d(j,1),int.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',6);
hold on;
            else

plot(int.d(j,1),int.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',6);
hold on;
            end
        end
        %plot the decision surface
        if rh~=-1
            delete(rh);
            delete(rh2);
            delete(rh3);
        end
        rh = annotation('rectangle',dsxy2figxy([[0 0] int.db(1)
int.db(2)])); hold on;
        rh2 = annotation('rectangle',dsxy2figxy([[0 0] int.dbUB(1)
int.dbUB(2)])); hold on;
        rh3 = annotation('rectangle',dsxy2figxy([[0 0] int.dbLB(1)
int.dbLB(2)])); hold on;
        set(rh,'LineStyle','--','EdgeColor','k','LineWidth',2);
        set(rh2,'LineStyle','--','EdgeColor','b','LineWidth',2);
        set(rh3,'LineStyle','--','EdgeColor','r','LineWidth',2);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = int.scale(1)*(xsTestSys(i,1)-int.shift(1));
        weightTestSys(i) = int.scale(2)*(xsTestSys((i-1)*100+1,2)-
int.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_subsys_Intervals()

    %halton sequence at the systems level
    %rule-based intervals

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations

    figure();
    ah = axes();
    rh = -1;

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems interval classifier
    intsys = systemsInt0(struct([]),100);

    int = struct([]); %start with the "empty" interval structure
    for i=1:I
        int = subsystemsInt0(int,M,intsys);

        cla(ah);
        axis([0 1 0 1]);
        %plot the data points
        for j=1:int.N
            yj = (int.d(j,3:4)-intsys.shift).*intsys.scale;
            if yj(1)<=intsys.db(1) && yj(2)<=intsys.db(2)
```

```
plot(int.d(j,1),int.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',6);
hold on;
            else

plot(int.d(j,1),int.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',6);
hold on;
            end
        end
        %plot the decision surface
        if int.dbUB(1)>int.dbLB(1) && int.dbUB(2)>int.dbLB(2)
            if rh~=-1
                delete(rh);
            end
            rh = annotation('rectangle',dsxy2figxy([int.dbLB
int.dbUB(1)-int.dbLB(1) int.dbUB(2)-int.dbLB(2)])); hold on;
            set(rh,'LineStyle','--','EdgeColor','k','LineWidth',2);
        end
        pause();
    end

    %plot the correct decision boundary
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = int.scale(1)*(xsTestSol(i,1)-int.shift(1));
        chordTestSol(i) = int.scale(2)*(xsTestSol((i-1)*100+1,2)-
int.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

end


function demo_sys03()

    %halton sequence at the systems level
    %rule-based sigmas, loss factors at systems level
    %fully connected BN

    I = 1; %repeat I times
    M = 100; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys,DinTestSys,DoutTestSys,NtotTestSys] =
dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = systemsKBN03(n,M);

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

%surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

%surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function experiment01()

    %Systems and Subsystems:
    %  halton sequence
    %  rule-based sigmas
    %  parzen window

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp01_se1.csv','w');   %false positives for systems
    fhs2 = fopen('exp01_se2.csv','w');   %false negatives for systems
    fhss1 = fopen('exp01_sse1.csv','w'); %false positives for
subsystems
```

```
    fhss2 = fopen('exp01_sse2.csv','w'); %false negatives for
subsystems

    %standard deviation files
    fhs3 = fopen('exp01_sh1.csv','w');   %acceptable standard
deviations for systems
    fhs4 = fopen('exp01_sh2.csv','w');   %unacceptable standard
deviations for systems
    fhss3 = fopen('exp01_ssh1.csv','w'); %acceptable standard
deviations for subsystems
    fhss4 = fopen('exp01_ssh2.csv','w'); %unacceptable standard
deviations for subsystems

    its = 1;
    while its<=1

        nsys = struct([]);
        nssys = struct([]);

        %systems classification
        for i=1:N
            [nsys] = systemsKBN01(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
            fprintf(fhs3,'%g,',nsys.h(1));
            fprintf(fhs4,'%g,',nsys.h(2));
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');
        fprintf(fhs3,'\n');
        fprintf(fhs4,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN01(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
            fprintf(fhss3,'%g,',nssys.h(1));
            fprintf(fhss4,'%g,',nssys.h(2));
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');
        fprintf(fhss3,'\n');
        fprintf(fhss4,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhs3);
    fclose(fhs4);
    fclose(fhss1);
    fclose(fhss2);
    fclose(fhss3);
    fclose(fhss4);

end


function experiment02()

    %Systems and Subsystems:
    %  halton sequences
    %  rule-based sigmas
    %  naive bayes

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp02_se1.csv','w');   %false positives for systems
    fhs2 = fopen('exp02_se2.csv','w');   %false negatives for systems
    fhss1 = fopen('exp02_sse1.csv','w'); %false positives for
subsystems
    fhss2 = fopen('exp02_sse2.csv','w'); %false negatives for
subsystems

    its = 1;
    while its<=1

        nsys = struct([]);
        nssys = struct([]);

        %systems classification

        for i=1:N
            [nsys] = systemsKBN02(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN02(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);

end


function experimentInt0()

    %Systems and Subsystems:
    %  halton sequence
    %  intervals

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,~,~,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,~,~,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('expInt.075_se1.csv','a');   %false positives for
systems
    fhs2 = fopen('expInt.075_se2.csv','a');   %false negatives for
systems
    fhss1 = fopen('expInt.075_sse1.csv','a'); %false positives for
subsystems
    fhss2 = fopen('expInt.075_sse2.csv','a'); %false negatives for
subsystems

    its = 1;
    while its<=1

        intsys = [];
        intssys = [];

        %systems classification
        for i=1:N
            [intsys] = systemsInt0(intsys,1);
            [e1 e2] = intGetErr1(intsys,xTestS,cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        for i=1:N
            [intssys] = subsystemsInt0(intssys,1,intsys);
            [e1 e2] = intGetErr2(intssys,xTestSS,cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end

    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);

    function [err1 err2] = intGetErr1(n,xs,cs)
        xs = [n.scale(1)*(xs(:,1)-n.shift(1)) n.scale(2)*(xs(:,2)-
n.shift(2))];
        Nerr = size(xs,1);
        nerr1 = 0;
        nerr2 = 0;
        for ii=1:Nerr
            if xs(ii,1)<=n.db(1) && xs(ii,2)<=n.db(2)
```

```
                if cs(ii,1)~=1
                    nerr1 = nerr1+1;
                end
            elseif cs(ii,1)~=2
                nerr2 = nerr2+1;
            end
        end
        err1 = nerr1/Nerr;
        err2 = nerr2/Nerr;
    end
    function [err1 err2] = intGetErr2(n,xs,cs)
        xs = [n.scale(1)*(xs(:,1)-n.shift(1)) n.scale(2)*(xs(:,2)-
n.shift(2))];
        Nerr = size(xs,1);
        nerr1 = 0;
        nerr2 = 0;
        for ii=1:Nerr
            if xs(ii,1)<=n.dbUB(1) && xs(ii,1)>=n.dbLB(1) &&
xs(ii,2)<=n.dbUB(2) && xs(ii,2)>=n.dbLB(2)
                if cs(ii,1)~=1
                    nerr1 = nerr1+1;
                end
            elseif cs(ii,1)~=2
                nerr2 = nerr2+1;
            end
        end
        err1 = nerr1/Nerr;
        err2 = nerr2/Nerr;
    end

end


function experiment03()

    %Systems and Subsystems:
    %  halton sequence
    %  rule-based sigmas, loss factors at systems level
    %  fully connected BN

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp03_se1.csv','w');   %false positives for systems
    fhs2 = fopen('exp03_se2.csv','w');   %false negatives for systems
    fhss1 = fopen('exp03_sse1.csv','w'); %false positives for
subsystems
    fhss2 = fopen('exp03_sse2.csv','w'); %false negatives for
subsystems

    its = 1;
    while its<=1

        nsys = struct([]);
        nssys = struct([]);

        %systems classification
        for i=1:N
            [nsys] = systemsKBN03(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN01(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);
end
```

Appendix D

The Matlab® code for Chapter 5's UAV solution 3 which investigates different methods for setting the standard deviations is presented in this appendix.

Table D.1: Matlab® Functions for UAV Solution 3

kbnEvalHadapt( )	The implementation of the proposed adaptive standard deviations method.
kbnEvalHadaptLOOCV( )	Minimization of the LOOCV error estimate for setting the standard deviations.
systemsKBN04( )	Adaptive standard deviations.
subsystemsKBN04( )	Adaptive standard deviations.
demo_sys04a( )	Plots runs of systemsKBN04, e.g. Fig. 5.10 left.
demo_subsys04a( )	Plots runs of subsystemsKBN04, e.g. Fig. 5.10 right.
systemsKBN00( )	Scott's rule for standard deviations.
subsystemsKBN00( )	Scott's rule for standard deviations.
experiment04( )	Generates errors and standard deviations. The same file was used to run all versions of systemsKBN0X and subsystemsKBN0X necessary to produce Fig. 5.2-9.

```
function h = kbnEvalHadapt(n,confGoal)

    withErrPlot = false;
    if withErrPlot
        fh2 = figure();
        ah2 = axes();
    end

    if n.Nc(1)==0 || n.Nc(2)==0 || (n.Nc(1)==1 && n.Nc(2)==1)
        h = kbnEvalH(n);
        return;
    end

    hLB = .01/(n.N^(1/n.Din));
    hUB = 2/(n.N^(1/n.Din));

    Pc1 = (n.Nc(1)+1)/(n.N+2);
    Pc2 = (n.Nc(2)+1)/(n.N+2);
    if withErrPlot
        figure(fh2);
        cla(ah2);
        errplotting = true;
        errPlot(@err,[hLB;hUB],(hUB-hLB)/100);
    end

    errplotting = false;
    hOpt = linesearch(@err,hLB,hUB);
    h = hOpt*ones(1,n.Din);

    function [errh] = err(hIn)
        dplus = 0;
        dminus = 0;
        n.h = hIn*ones(1,n.Din);
        for ii=1:n.N
            [pTemp] = kbnEval(n,n.d(ii,1:n.Din));
            pDiff = (Pc1*pTemp(1,1,1)-
Pc2*pTemp(1,1,2))/(Pc1*pTemp(1,1,1)+Pc2*pTemp(1,1,2));
            if n.w(ii,1)>0
                if pDiff<=confGoal
                    dminusii = confGoal-pDiff;
                    dplusii = 0.0;
                else
                    dminusii = 0.0;
                    dplusii = 0.0;
                end
            else
                if pDiff>=-confGoal
                    dminusii = 0.0;
                    dplusii = pDiff+confGoal;
                else
                    dminusii = 0.0;
                    dplusii = 0.0;
                end
            end
            dplus = dplus+dplusii;
            dminus = dminus+dminusii;
        end
        errh = dplus^2+dminus^2;
        if ~errplotting && withErrPlot
           figure(fh2);
           axis(ah2);
           plot(hIn,errh,'o'); hold on;
           %pause();
        end
    end

    function errPlot(errFN,limits,res)
        widths = limits(2,:)-limits(1,:);
        divs = ceil(widths./res);
        step = widths./divs;
        x1 = limits(1,1):step(1):limits(2,1);
        x1Count = divs(1)+1;
        y = zeros(1,x1Count);
        for ii=1:x1Count
            y(ii) = errFN(x1(ii));
        end
        axis(ah2);
        plot(x1,y); hold on;
        xlim(limits(:,1)');
    end

    function [x] = linesearch(f,xl,xu)
        %bracket the onset of error
        f0 = 1;
        while f0>0
            x0 = xl;
            f0 = f(x0);
            xl = xl/2;
        end
        dx = .01;
        x1 = x0+dx;
        f1 = f(x1);
        while f1<=f0
            x0 = x1;
            f0 = f1;
            dx = 1.5*dx;
            x1 = x0+dx;
            if x1>=xu
                x = xu;
                return;
            end
            f1 = f(x1);
        end

        %tighten bracket using golden section
        lb = x0;
        ub = x1;
        gs = (sqrt(5)-1)/2;
        ux = lb+gs*(ub-lb);
        lx = ub-gs*(ub-lb);
        uf = f(ux);
        lf = f(lx);
        while (ub-lb)>.0001
            %sprintf('lb = %.4g lbf = %.4g ub = %.4g ubf =
%.4g',lb,lbf,ub,ubf)
            if (uf<=lf)
                lb = lx;
                lx = ux;
                lf = uf;
                ux = lb + gs*(ub-lb);
                uf = f(ux);
            else
                ub = ux;
                ux = lx;
                uf = lf;
                lx = ub - gs*(ub-lb);
                lf = f(lx);
            end
        end

        %return the best answer
        x = lb;
    end

end


function h = kbnEvalHadaptLOOCV(n)

    if n.Nc(1)<2 || n.Nc(2)<2
        h = zeros(1,n.C);
        for i=1:n.C
            if n.Nc(i)+n.hshift<=0
                h(i) = n.hscale(i);
            else
                h(i) = n.hscale(i)/(n.Nc(i)+n.hshift(i))^(1/n.Din);
            end
        end
        return;
    end

    withErrPlot = false;
    if withErrPlot
        fh2 = figure();
        ah2 = axes();
    end

    hLB = .01/(n.N^(1/n.Din));
    hUB = 2/(n.N^(1/n.Din));

    if withErrPlot
        figure(fh2);
        cla(ah2);
        errplotting = true;
        errPlot(@err,[hLB;hUB],(hUB-hLB)/100);
    end
    errplotting = false;

    h0 = (hUB+hLB)/2;
    hOpt = linesearch(@err,h0,hLB,hUB);
    h = hOpt*ones(1,n.Din);
    function [errh] = err(hIn)
        errh = 0;
        ncv = kbn(n.Din,n.Dout,2);
        ncv.N = n.N-1;
        ncv.h = hIn*ones(1,n.Din);
        for ii=1:n.N
            ncv.d = [n.d(1:ii-1,:); n.d(ii+1:n.N,:)];
            ncv.w = [n.w(1:ii-1,:); n.w(ii+1:n.N,:)];
            ncv.Nc(1) = nnz(ncv.w(:,1));
            ncv.Nc(2) = nnz(ncv.w(:,2));
            ncv.w(:,1) = ncv.w(:,1)*n.Nc(1)/ncv.Nc(1);
            ncv.w(:,2) = ncv.w(:,2)*n.Nc(2)/ncv.Nc(2);
            Pc1 = (ncv.Nc(1)+1)/(ncv.N+2);
            Pc2 = (ncv.Nc(2)+1)/(ncv.N+2);
            errii = 0;
            [pTemp] = kbnEval(ncv,n.d(ii,1:n.Din));
            pDiff = (Pc1*pTemp(1,1,1)-
Pc2*pTemp(1,1,2))/(Pc1*pTemp(1,1,1)+Pc2*pTemp(1,1,2));
            %pDiff = Pc1*pTemp(1,1,1)-Pc2*pTemp(1,1,2);
            if pDiff>=0
                if n.w(ii,1)>0
                    errii = errii-pDiff/(1*n.Nc(1));
                else
                    errii = errii+pDiff;
                end
            else
                if n.w(ii,1)>0
                    errii = errii-pDiff;
               else
                    errii = errii+pDiff/(1*n.Nc(2));
                end
            end
            errh = errh+errii;
        end
        errh = errh/n.N;
        if ~errplotting && withErrPlot
           figure(fh2);
           axis(ah2);
           plot(hIn,errh,'o'); hold on;
           %pause();
        end
    end

    function errPlot(errFN,limits,res)
        widths = limits(2,:)-limits(1,:);
        divs = ceil(widths./res);
        step = widths./divs;
        x1 = limits(1,1):step(1):limits(2,1);
        x1Count = divs(1)+1;
        y = zeros(1,x1Count);
        for ii=1:x1Count
            y(ii) = errFN(x1(ii));
        end
        axis(ah2);
        plot(x1,y); hold on;
        xlim(limits(:,1)');
    end

    function [x] = linesearch(f,x0,xl,xu)
        %bracket the minimum error
        f0 = f(x0);
        dx = .01;
        x1 = x0+dx;
        f1 = f(x1);
        if f1>f0
            x00 = x1;
            dx = -.01;
            x1 = x0+dx;
            f1 = f(x1);
        end

        while f1<=f0
            x00 = x0;
            x0 = x1;
            f0 = f1;
            dx = 1.5*dx;
            x1 = x0+dx;
            if x1>=xu
                if x0==xu
                    x = xu;
                    return;
                end
                x1 = xu;
            end
            if x1<=xl
                if x0==xl
                    x = xl;
                    return;
                end
                x1 = xl;
            end
            f1 = f(x1);
        end

        %tighten bracket using golden section
        if dx>0
            lb = x00;
            ub = x1;
        else
            lb = x1;
            ub = x00;
        end
        gs = (sqrt(5)-1)/2;
        ux = lb+gs*(ub-lb);
```

```
        lx = ub-gs*(ub-lb);
        uf = f(ux);
        lf = f(lx);
        while (ub-lb)>.0001
            %sprintf('lb = %.4g lbf = %.4g ub = %.4g ubf =
%.4g',lb,lbf,ub,ubf)
            if (uf<=lf)
                lb = lx;
                lx = ux;
                lf = uf;
                ux = lb + gs*(ub-lb);
                uf = f(ux);
            else
                ub = ux;
                ux = lx;
                uf = lf;
                lx = ub - gs*(ub-lb);
                lf = f(lx);
            end
        end

        %return the best answer
        x = (lb+ub)/2;
    end

end


function n = systemsKBN04(n,M,c)

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are adaptive.

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end
    end

    %set the standard deviation
    n.h = kbnEvalHadapt(n,c); %adaptive

end


function [n] = subsystemsKBN04(n,M,nsys,c)

    classifyType = 0;
    % 1 for using the systems classifier
    % otherwise use the systems simulation

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are adaptive.

    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4]);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        if classifyType
            %classify the point using systems' classifier
            [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
            if ci==0
                ci = 2;
            end
        else
            %or classify correctly using the systems' simulation
            range = systemsF0(yi);
            if range>=900
                ci = 1;
            else
                ci = 2;
            end
        end
        n = kbnAddData(n,[xi yi],ci);

    end

    %set the standard deviation
    n.h = kbnEvalHadapt(n,c); %adaptive

end


function demo_sys04a()

    %halton sequence
    %adaptive sigmas
    %fully connected BN

    M = 10; %the number of data points each time

    figh = figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys,DinTestSys,DoutTestSys,NtotTestSys] =
dataRead(fh);
    fclose(fh);

    c = [0 .5 1];
    cc{1} = 'g--';
    cc{2} = 'b-';
    cc{3} = 'k:';
    for i=1:length(c)
        n = systemsKBN04(struct([]),M,c(i));

        figure(figh);
        %cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

%surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

%surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],cc{i},'LineWidth',2); hold
on;
        axis([0 1 0 1]);
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',5); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',10);
hold on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',10);
hold on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',5); hold
on;
                end
            end
        end
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_subsys04a()

    %halton sequence
    %adaptive sigmas
    %fully connected BN

    M = 10; %the number of data points each time
    figh = figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol,~,~,NtotTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN04(struct([]),100,.5);

    c = [0 .5 1];
    cc{1} = 'g--';
    cc{2} = 'b-';
    cc{3} = 'k:';
    for i=1:length(c)
        n = subsystemsKBN04(struct([]),M,nsys,c(i));

        figure(figh);
        %cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

%surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

%surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],cc{i},'LineWidth',2); hold
on;
        axis([0 1 0 1]);
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',5); hold
on;
                else
```

```
plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',10);
hold on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',10);
hold on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',5); hold
on;
                end
            end
        end
        pause();
    end

    %plot the correct decision boundary
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

end


function n = systemsKBN00(n,M)

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based: scott's rule

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = ones(1,n.C)/sqrt(12);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation using scott's rule
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/(n.Din+4)));
        end
    end

end


function [n] = subsystemsKBN00(n,M,nsys)

    classifyType = 0;
    % 1 for using the systems classifier
    % otherwise use the systems simulation

    %The search strategy is exploratory only using a halton sequence.
    %The sigmas are rule-based.

    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        n.hscale = ones(1,n.C)/sqrt(12);
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        if classifyType
            %classify the point using systems' classifier
            [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
            if ci==0
                ci = 2;
            end
        else
            %or classify correctly using the systems' simulation
            range = systemsF0(yi);
            if range>=900
                ci = 1;
            else
                ci = 2;
            end
        end
        n = kbnAddData(n,[xi yi],ci);

    end

    %set the standard deviation using scott's rule
    n.h = zeros(1,n.C);
    for i=1:n.C
        if n.Nc(i)<=0
            n.h(i) = n.hscale(i);
        else
            n.h(i) = n.hscale(i)/(n.Nc(i)^(1/(n.Din+4)));
        end
    end

end


function experiment04()

    %Systems and Subsystems:
    %  halton sequence
    %  adaptive sigmas
    %  fully connected BN

    %1.0 halton at systems for 100 points
    %2.0 halton at subsystems for 100 points using systems bn
    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);
    %error output files
    fhs1 = fopen('exp04_se1.csv','a');   %false positives for systems
    fhs2 = fopen('exp04_se2.csv','a');   %false negatives for systems
    fhss1 = fopen('exp04_sse1.csv','a'); %false positives for
subsystems
    fhss2 = fopen('exp04_sse2.csv','a'); %false negatives for
subsystems

    %standard deviation files
    fhs3 = fopen('exp04_sh1.csv','a');   %acceptable standard
deviations for systems
    fhs4 = fopen('exp04_sh2.csv','a');   %unacceptable standard
deviations for systems
    fhss3 = fopen('exp04_ssh1.csv','a'); %acceptable standard
deviations for subsystems
    fhss4 = fopen('exp04_ssh2.csv','a'); %unacceptable standard
deviations for subsystems

    its = 1;
    while its<=1

        nsys = [];
        nssys = [];

        %systems classification
        for i=1:N
            [nsys] = systemsKBN04(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
            fprintf(fhs3,'%g,',nsys.h(1));
            fprintf(fhs4,'%g,',nsys.h(2));
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');
        fprintf(fhs3,'\n');
        fprintf(fhs4,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN04(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
            fprintf(fhss3,'%g,',nssys.h(1));
            fprintf(fhss4,'%g,',nssys.h(2));
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');
        fprintf(fhss3,'\n');
        fprintf(fhss4,'\n');

        its = its+1;
    end

    fclose(fhs1);
    fclose(fhs2);
    fclose(fhs3);
    fclose(fhs4);
    fclose(fhss1);
    fclose(fhss2);
    fclose(fhss3);
    fclose(fhss4);

end
```

Appendix E

The Matlab® code for Chapter 6's UAV solutions 4 and 5 for incorporating expert knowledge is presented in this appendix.

Table E.1: Matlab® Functions for UAV Solutions 4 and 5

systemsKBN05( )	Expert knowledge from seeded training points.
systemsKBN06( )	Expert knowledge from automatic classification.
demo_sys05( )	Plots runs of systemsKBN05, e.g. Fig. 6.1.
demo_sys06( )	Plots runs of systemsKBN06, e.g. Fig. 6.3.
experiment05( )	Generates errors for systemsKBN05, e.g. Fig. 6.2.
experiment06( )	Generates errors for systemsKBN06, e.g. Fig. 6.4.

```
function n = systemsKBN05(n,M)

    %The search strategy is exploratory only using a halton sequence.
    %Expert knowledge is embedded.
    %The sigmas are rule-based.

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = 0.4*ones(1,n.Din);

        %expert input
        Nexp0 = 10;
        Nexp1 = 10;
        expd = zeros(Nexp0+Nexp1,n.Din+n.Dout);
        expd(Nexp0+1:Nexp0+Nexp1,1:n.Din) = ones(Nexp1,n.Din);
        expc = ones(Nexp0+Nexp1);
        expc(Nexp0+1:Nexp0+Nexp1) = 2*ones(1,Nexp0);
        n = kbnAddData(n,expd,expc);
        n.hshift = -Nexp0-Nexp1;
    end

    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+n.hshift+1);
        xi = xhalton(n.N+n.hshift+1,:);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation
    n.h = kbnEvalH(n); %rule-based

end


function [n Nf] = systemsKBN06(n,M)

    %The search strategy is exploratory only using a halton sequence.
    %Monotonicity is used at the systems level.
    %The sigmas are rule-based.

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = 0.4*ones(1,n.Din);
        n.dom1 = []; %the dominating set for class 1
        n.dom2 = []; %the dominating set for class 2
    end

    Nf = 0;
    for i=1:M

        %get the next data point
        xhalton = halton(n.Din,n.N+1);
        xi = xhalton(n.N+1,:);

        %check for monotonic dominance
        class = 0;
        yi = 0;
        if n.N>0
            for j=1:size(n.dom1,1)
                if xi(1)<=n.dom1(j,1) && xi(2)<=n.dom1(j,2)
                    class = 1;
                    break;
                end
            end
            if class==0
```

```
                for j=1:size(n.dom2,1)
                    if xi(1)>=n.dom2(j,1) && xi(2)>=n.dom2(j,2)
                        class = 2;
                        break;
                    end
                end
            end
        end
        if class==0 || n.N==0
            Nf = Nf + 1;
            yi = systemsF0((xi./n.scale+n.shift));
            if yi>=rangeDB
                class = 1;
                dj = [];
                for j=1:size(n.dom1,1)
                    if xi(1)>n.dom1(j,1) && xi(2)>n.dom1(j,2)
                        dj = [dj; j];
                    end
                end
                n.dom1(dj,:) = [];
                if isempty(n.dom1)
                    n.dom1 = [xi yi];
                else
                    n.dom1 = [n.dom1;xi yi];
                end
            else
                class = 2;
                dj = [];
                for j=1:size(n.dom2,1)
                    if xi(1)<n.dom2(j,1) && xi(2)<n.dom2(j,2)
                        dj = [dj; j];
                    end
                end
                n.dom2(dj,:) = [];
                if isempty(n.dom2)
                    n.dom2 = [xi yi];
                else
                    n.dom2 = [n.dom2;xi yi];
                end
            end
        end

        %classify the point
        n = kbnAddData(n,[xi yi],class);
    end

    %set the standard deviation
    n.h = kbnEvalH(n); %rule-based

end
```

```
function demo_sys05()

    %halton sequence w/ expert input at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys] = dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    for i=1:I+1
        if i==1
            n = systemsKBN05(n,0);
        else
            n = systemsKBN05(n,M);
        end

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
```

```
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_sys06()

    %halton sequence w/ monotonicity input at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations

    figure();
    ah = axes();
    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys] = dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    Ns = 0;
    for i=1:I
        [n Nsi] = systemsKBN06(n,M);
        Ns = Ns+Nsi

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',6); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',6); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',6); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',6); hold
on;
                end
            end
        end
        for j=1:size(n.dom1,1)
```

```
plot(n.dom1(j,1),n.dom1(j,2),'b>','MarkerFaceColor','b','MarkerSize',12
); hold on;
        end
        for j=1:size(n.dom2,1)

plot(n.dom2(j,1),n.dom2(j,2),'r<','MarkerFaceColor','r','MarkerSize',12
); hold on;
        end

        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function experiment05()

    %Systems and Subsystems:
    %  halton sequence w/ expert input at systems level
    %  rule-based sigmas
    %  fully connected BN

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp05_se1.csv','w');   %false positives for systems
    fhs2 = fopen('exp05_se2.csv','w');   %false negatives for systems
    fhss1 = fopen('exp05_sse1.csv','w'); %false positives for
subsystems
    fhss2 = fopen('exp05_sse2.csv','w'); %false negatives for
subsystems

    its = 1;
    while its<=1

        nsys = struct([]);
        nssys = struct([]);

        %systems classification
        for i=1:N
            [nsys] = systemsKBN05(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN10(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);

end
```

```
function experiment06()

    %Systems and Subsystems:
    %  halton sequence w/ monotonicity input at systems level
    %  rule-based sigmas
    %  fully connected BN

    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp06_se1.csv','w');   %false positives for systems
    fhs2 = fopen('exp06_se2.csv','w');   %false negatives for systems
    fhs3 = fopen('exp06_sns.csv','w');   %function calls for systems
    fhss1 = fopen('exp06_sse1.csv','w'); %false positives for
subsystems
    fhss2 = fopen('exp06_sse2.csv','w'); %false negatives for
subsystems

    its = 1;
    while its<=1

        nsys = struct([]);
        nssys = struct([]);

        %systems classification
        Ns = 0;
        for i=1:N
            [nsys Nsi] = systemsKBN06(nsys,1);
            Ns = Ns+Nsi;
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
            fprintf(fhs3,'%d,',Ns);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');
        fprintf(fhs3,'\n');

        %subsystems classification
        for i=1:N
            [nssys] = subsystemsKBN10(nssys,1,nsys);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end

    fclose(fhs1);
    fclose(fhs2);
    fclose(fhs3);
    fclose(fhss1);
    fclose(fhss2);

end
```

Appendix F

The Matlab® code for Chapter 7's UAV solutions 6-8 for adaptive sampling is presented in this appendix.

Table F.1: Matlab® Functions for UAV Solutions 6-8

kbnSampleExplore( )	Determine the next exploratory samples using the exploratory adaptive sampling method
kbnEvalHExplore( )	Calculate the standard deviation for the exploratory KDE.
kbnSample( )	Sample from a KBN for exploitive adaptive sampling.
systemsKBN11( )	Exploratory adaptive sampling.
subsystemsKBN11( )	Exploratory and search domain exploitive adaptive sampling.
subsystemsKBN12( )	Exploratory, search domain exploitive, and feasible region exploitive adaptive sampling.
demo_sys11( )	Plots runs of systemsKBN11, e.g. Fig. 7.3.
demo_subsys11( )	Plots runs of subsystemsKBN11, e.g. Fig. 7.5, 7.8, 7.9.
demo_subsys12( )	Plots runs of subsystemsKBN12, e.g. Fig. 7.13.
experiment11( )	Generates errors for systemsKBN11, subsystemsKBN11, e.g. Fig. 7.4, 7.6, 7.10.
experiment12( )	Generates errors for systemsKBN11, subsystemsKBN12, e.g. Fig. 7.14.

```
function [xs N] = kbnSampleExplore(n,M)

    %use multistart SQP to find the mins of the full joint
    itsConvMax = 5;
    percentConv = .01;
    timeMax = .1;

    D = n.Din;
    N = n.N;
    if N==0
        xs = zeros(M,D);
        xs(1,:) = rand(1,D);
        N = 1;
        if M==1
            return;
        else
```

```
            M=M-1;
        end
    else
        xs = [n.d(:,1:D); zeros(M,D)];
    end

    options = optimset('GradObj','on','Display','off');
    for iN=N+1:N+M
        s = kbnEvalHexplore(D,iN-1);
        its = 0;
        time = 0;
        while 1
            tic();
            its = its + 1;
            x0 = rand(1,D);
            [xtest ptest] =
fmincon(@evalP,x0,[],[],[],[],zeros(1,D),ones(1,D),[],options);
            if its==1
                pmin = ptest;
                xmin = xtest;
                pminBL = pmin;
                itsConv = 1;
            elseif ptest<pmin
                pmin = ptest;
                xmin = xtest;
                if (pminBL-pmin)/pminBL < percentConv
                    itsConv = itsConv + 1;
                else
                    itsConv = 1;
                    pminBL = pmin;
                end
            else
                itsConv = itsConv + 1;
            end
            time = time + toc();
            if itsConv>=itsConvMax || time>=timeMax
                %if time>=timeMax
                %    display('Max time reached.');
                %end
                break;
            end
        end
        xs(iN,:) = xmin;
    end
    N = n.N+M;
    xs = xs(n.N+1:N,:);

    function [p dp] = evalP(xx)
        p = 0;
        dp = zeros(1,D);
        dpp = zeros(1,D);
        for ii=1:iN-1
```

```
            pp = 1.0;
            for jj=1:D;
                %pp = pp*exp(-((xx(jj)-
xs(ii,jj))^2)/(2*s(jj)^2))/(s(jj)*sqrt(2*pi));
                pp = pp*exp(-((xx(jj)-xs(ii,jj))^2)/(2*s(jj)^2));
            end
            for jj=1:D;
                dpp(jj) = -(xx(jj)-xs(ii,jj))*pp/(s(jj)^2);
                dp(jj) = dp(jj) + dpp(jj);
            end
            p = p + pp;
        end
        %dp = dp/(iN-1);
        %p = p/(iN-1);
    end

end


function [h] = kbnEvalHexplore(D,N)

    if N<=1
        N = 2;
    end
    h = (.425*((D/(D+1))^.5)/(N^(1/D)-1))*ones(1,D);

end


function [xs N] = kbnSample(n,M,c)

   %sample a KBN in ancestral order for M designs

   D = n.Din;
   N = n.Nc(c);
   di = find(n.w(:,c));
   d = n.d(di,:);
   w = n.w(di,:);
   h = n.h(c);
   xs = NaN.*ones(M,D);
   l = 0;
   while l < M
       xk = NaN.*ones(1,D);
       pjk = NaN.*ones(N,D);
       outofbounds = false;
       for k=1:D
           xtest = sample(k);
           if xtest<0 || xtest>1
               outofbounds = true;
               break;
           else
```

```
                xk(k)=xtest;
            end
        end
        if ~outofbounds
            l = l+1;
            xs(l,:) = xk;
        end
    end
    N = n.N+M;

    function x = sample(vk)

        vkp = n.g{vk}.p;
        anum = ones(1,N);
        aden = 1;
        if ~isempty(vkp)
            aden = 0;
            for jk=1:N
                for kk=1:length(vkp)
                    anum(jk) = anum(jk)*pjk(jk,vkp(kk));
                end
                aden = aden+w(jk)*anum(jk);
            end
        end

        %estimate the upper and lower ends of the PD
        xlo = min(d(:,vk))-6*h;
        xhi = max(d(:,vk))+6*h;

        %find a reasonable resolution based on the standard deviation
        res = h/10;
        xdiv = ceil((xhi-xlo)/res);
        if xdiv>10000
            xdiv = 10000;
        end
        xstep = (xhi-xlo)/xdiv;

        %build the cumulative PD for vvi
        xks = zeros(1,xdiv+1);
        pks = zeros(1,xdiv+1);
        cpks = zeros(1,xdiv+1);
        r = rand();
        for ik=1:xdiv+1
            xks(ik) = xlo+(ik-1)*xstep;
            pks(ik) = 0.0;
            for jk=1:N
                pjk(jk,vk) = n.k(xks(1,ik),d(jk,vk),h);
                pks(ik) = pks(ik) + w(jk)*anum(jk)*pjk(jk,vk);
            end
            pks(ik) = pks(ik)/aden;
            if ik==1
                cpks(ik) = pks(ik)*xstep;
            else
                cpks(ik) = cpks(ik-1)+pks(ik)*xstep;
            end
            if r<=cpks(ik)
                x = xks(ik);
                break;
            end
        end
    end

end


function n = systemsKBN11(n,M)

    %The search strategy is KBN exploratory.
    %The sigmas are rule-based: new rule

    rangeDB = 900; %acceptable range lower bound
    if isempty(n)
        n = kbn(2,1,2,'bnd',[0 0;1 5]);
        n.hscale = 0.4*ones(1,n.Din);
    end

    for i=1:M

        %get the next data point
        xi = kbnSampleExplore(n,1);
        yi = systemsF0((xi./n.scale+n.shift));

        %classify the point
        if yi>=rangeDB
            n = kbnAddData(n,[xi yi],1);
        else
            n = kbnAddData(n,[xi yi],2);
        end

    end

    %set the standard deviation using the new rule
    n.h = kbnEvalH(n);

end


function [n, nn] = subsystemsKBN11(n,nn,M,nsys,alpha)

    %This function classifies the subsystem's calculation of drag and
weight
    %  according to the systems level classifier
    %The search strategy is exploratory/exploitive.
    %The sigmas are rule-based.

    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        n.hscale = 0.4*ones(1,n.Din);
    end
    if isempty(nn)
        nn = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        nn.hscale = ones(1,n.Din);
    end

    for i=1:M

        %get the next data point
        rn = rand();
        if n.N<10
            %exploratory
            xi = kbnSampleExplore(n,1);
        elseif nn.Nc(1)>0 && rn<alpha
            %from the systems' search domain
            xi = kbnSample(nn,1,1);
        else
            %exploratory
            xi = kbnSampleExplore(n,1);
        end
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        %classify the point using systems' classifier
        [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
        if ci(1)==0
            nn = kbnAddData(nn,[xi yi],2);
            ci(1) = 2;
        else
            nn = kbnAddData(nn,[xi yi],1);
        end
        n = kbnAddData(n,[xi yi],ci(1));
        nn.h = kbnEvalH(nn);

    end

    %set the standard deviation using the new rule
    n.h = kbnEvalH(n);

end


function [n, nn] = subsystemsKBN12(n,nn,M,nsys,alpha)
```

```
    %The search strategy is exploratory/exploitive.
    %The sigmas are rule-based.

    if isempty(n)
        n = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        n.hscale = 0.4*ones(1,n.Din);
    end
    if isempty(nn)
        nn = kbn(2,2,2,'bnd',[1 .1;15 .4]);
        nn.hscale = ones(1,n.Din);
    end

    for i=1:M

        %get the next data point
        rn = rand(1,2);
        if n.N<10
            %exploratory
            xi = kbnSampleExplore(n,1);
        elseif n.Nc(1)>0 && rn(1)<alpha(1)
            %from the systems' acceptable design space
            xi = kbnSample(n,1,1);
        elseif nn.Nc(1)>0 && rn(2)<alpha(2)
            %from the systems' search domain
            xi = kbnSample(nn,1,1);
        else
            %exploratory
            xi = kbnSampleExplore(n,1);
        end
        yi(1) = aeroF0((xi./n.scale+n.shift));
        yi(2) = structF0((xi./n.scale+n.shift));

        %classify the point using systems' classifier
        [ci] = kbnEvalC(nsys,(yi-nsys.shift).*nsys.scale);
        if ci(1)==0
            nn = kbnAddData(nn,[xi yi],2);
            ci(1) = 2;
        else
            nn = kbnAddData(nn,[xi yi],1);
        end
        n = kbnAddData(n,[xi yi],ci(1));
        n.h = kbnEvalH(n);
        nn.h = kbnEvalH(nn);

    end

end


function demo_sys11()
```

```
    %halton sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations

    figure();
    ah = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsystemsN10000.csv');
    [xsTestSys,ysTestSys] = dataRead(fh);
    fclose(fh);

    n = struct([]); %start with the "empty" bn
    for i=1:I
        n = systemsKBN11(n,M);

        cla(ah);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1, ph1, dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0
```

```
plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        pause();
    end

    %plot the correct decision boundary
    dragTestSys = zeros(1,100);
    weightTestSys = zeros(1,100);
    rangeTestSys = zeros(100,100);
    for i=1:100
        dragTestSys(i) = n.scale(1)*(xsTestSys(i,1)-n.shift(1));
        weightTestSys(i) = n.scale(2)*(xsTestSys((i-1)*100+1,2)-
n.shift(2));
        rangeTestSys(i,:) = ysTestSys((i-1)*100+1:i*100,1);
    end
    contour(dragTestSys,weightTestSys,rangeTestSys,[900
900],'k','LineWidth',2); hold on;

end


function demo_subsys11()

    %halton sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations

    fhn = figure();
    ahn = axes();
    fhnn = figure();
    ahnn = axes();
    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN11(struct([]),100);

    n = struct([]); %start with the "empty" bn
    nn = struct([]);
    for i=1:I
        [n nn] = subsystemsKBN11(n,nn,M,nsys,1);

        figure(fhn)
        cla(ahn);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);

        figure(fhnn)
        cla(ahnn);
        Pc1 = (nn.Nc(1)+1)/(nn.N+2);
        Pc2 = (nn.Nc(2)+1)/(nn.N+2);
        [xh1 ph1 dph1] = kbnPlot(nn,[.025 .025],[nn.lf(1)*Pc1
nn.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),nn.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),nn.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:nn.N
            pTemp = kbnEval(nn,nn.d(j,1:nn.Din));
            pDiff = (nn.lf(1)*Pc1*pTemp(1,1,1)-
nn.lf(2)*Pc2*pTemp(1,1,2));
            if nn.w(j,1)>0
                if pDiff>0

plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
                else

plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
                end
            else
                if pDiff>0

plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
                else

plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
                end
            end
        end
        %plot the decision surface
```

```
%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);

        pause();
    end

    %plot the correct decision boundary
    figure(fhn);
    axis(ahn);
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    dragTestSol = zeros(100,100);
    weightTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
        dragTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,1);
        weightTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,2);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

    figure(fhnn);
    axis(ahnn);
    contour(tocTestSol,chordTestSol,dragTestSol,[1
1],'k','LineWidth',2); hold on;
    contour(tocTestSol,chordTestSol,weightTestSol,[5
5],'k','LineWidth',2); hold on;

end


function demo_subsys12()

    %KBN exploratory sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations

    fhn = figure();
    ahn = axes();
    fhnn = figure();
    ahnn = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN11(struct([]),100);

    n = struct([]); %start with the "empty" bn
    nn = struct([]);
    for i=1:I
        [n nn] = subsystemsKBN12(n,nn,M,nsys,[0 1]);

        figure(fhn)
        cla(ahn);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else
```

```
plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);

        figure(fhnn)
        cla(ahnn);
        Pc1 = (nn.Nc(1)+1)/(nn.N+2);
        Pc2 = (nn.Nc(2)+1)/(nn.N+2);
        [xh1 ph1 dph1] = kbnPlot(nn,[.025 .025],[nn.lf(1)*Pc1
nn.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),nn.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),nn.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:nn.N
            pTemp = kbnEval(nn,nn.d(j,1:nn.Din));
            pDiff = (nn.lf(1)*Pc1*pTemp(1,1,1)-
nn.lf(2)*Pc2*pTemp(1,1,2));
            if nn.w(j,1)>0
                if pDiff>0

plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
                else

plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
                end
            else
                if pDiff>0

plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
                else

plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
```

```
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);

        pause();
    end

    %plot the correct decision boundary
    figure(fhn);
    axis(ahn);
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    dragTestSol = zeros(100,100);
    weightTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
        dragTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,1);
        weightTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,2);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;

    figure(fhnn);
    axis(ahnn);
    contour(tocTestSol,chordTestSol,dragTestSol,[1
1],'k','LineWidth',2); hold on;
    contour(tocTestSol,chordTestSol,weightTestSol,[5
5],'k','LineWidth',2); hold on;

end


function experiment11()

    %Systems and Subsystems:
    %  kbn exploratory sequence
    %  rule-based sigmas: new rule
    %  parzen window
    %1.0 kbn explore at systems for 100 points
    %2.0 kbn explore at subsystems for 100 points using systems bn
    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp11_se1.csv','a');   %false positives for systems
    fhs2 = fopen('exp11_se2.csv','a');   %false negatives for systems
    fhss1 = fopen('exp11_sse1.csv','a'); %false positives for
subsystems
    fhss2 = fopen('exp11_sse2.csv','a'); %false negatives for
subsystems
    fhssn1 = fopen('exp11_ssn1.csv','a'); %# acceptable to systems
    fhssn2 = fopen('exp11_ssn2.csv','a'); %# in systems search domain

    its = 1;
    itsMax = 200;
    while its<=itsMax

        nsys = struct([]);
        nssys = struct([]);
        nnssys = struct([]);

        %systems classification
        for i=1:N
            [nsys] = systemsKBN11(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        n1 = 0;
        n2 = 0;
        for i=1:N
            [nssys] = subsystemsKBN11(nssys,nnssys,1,nsys,0);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
            if nssys.w(i,1)>0
                range = systemsF0(nssys.d(i,3:4));
                if range>=900
                    n1 = n1+1;
                    n2 = n2+1;
                end
            elseif nssys.d(i,3)>=nsys.bnd(1,1) &&
nssys.d(i,3)<=nsys.bnd(2,1) && ...
                   nssys.d(i,4)>=nsys.bnd(1,2) &&
nssys.d(i,4)<=nsys.bnd(2,2)
                n2 = n2+1;
            end
            fprintf(fhssn1,'%g,',n1);
            fprintf(fhssn2,'%g,',n2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');
        fprintf(fhssn1,'\n');
        fprintf(fhssn2,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);
    fclose(fhssn1);
    fclose(fhssn2);

end


function experiment12()

    %Systems and Subsystems:
    %  kbn exploratory sequence
    %  rule-based sigmas: new rule
    %  parzen window

    %1.0 kbn explore at systems for 100 points
    %2.0 kbn explore at subsystems for 100 points using systems bn
    N = 100;

    %error test points for systems
    fh = fopen('dataTestsystemsN1000.csv');
    [xTestS,yTestS,DinTestS,DoutTestS,NtotTestS] = dataRead(fh);
    cTestS = 2*ones(NtotTestS,1);
    cTestS(yTestS>=900) = 1;
    fclose(fh);

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,DinTestSS,DoutTestSS,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhs1 = fopen('exp12_se1.csv','a');   %false positives for systems
    fhs2 = fopen('exp12_se2.csv','a');   %false negatives for systems
    fhss1 = fopen('exp12_sse1.csv','a'); %false positives for
subsystems
    fhss2 = fopen('exp12_sse2.csv','a'); %false negatives for
subsystems
    fhssn1 = fopen('exp12_ssn1.csv','a'); %# acceptable to systems
    fhssn2 = fopen('exp12_ssn2.csv','a'); %# in systems search domain

    its = 1;
    itsMax = 200;
    while its<=itsMax

        nsys = struct([]);
        nssys = struct([]);
        nnssys = struct([]);

        %systems classification
        for i=1:N
            [nsys] = systemsKBN11(nsys,1);
            [e1 e2] = kbnGetErr(nsys,xTestS,[1 2],cTestS);
            fprintf(fhs1,'%g,',e1);
            fprintf(fhs2,'%g,',e2);
        end
        fprintf(fhs1,'\n');
        fprintf(fhs2,'\n');

        %subsystems classification
        n1 = 0;
        n2 = 0;
        for i=1:N
            [nssys nnssys] = subsystemsKBN12(nssys,nnssys,1,nsys,[0
1]);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
            if nssys.w(i,1)>0
                range = systemsF0(nssys.d(i,3:4));
                if range>=900
                    n1 = n1+1;
                    n2 = n2+1;
                end
            elseif nssys.d(i,3)>=nsys.bnd(1,1) &&
nssys.d(i,3)<=nsys.bnd(2,1) && ...
                   nssys.d(i,4)>=nsys.bnd(1,2) &&
nssys.d(i,4)<=nsys.bnd(2,2)
                n2 = n2+1;
            end
            fprintf(fhssn1,'%g,',n1);
            fprintf(fhssn2,'%g,',n2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');
        fprintf(fhssn1,'\n');
        fprintf(fhssn2,'\n');

        its = its+1;
    end
    fclose(fhs1);
    fclose(fhs2);
    fclose(fhss1);
    fclose(fhss2);
    fclose(fhssn1);
    fclose(fhssn2);

end
```

Appendix G

The Matlab® code for Chapter 8's UAV solution 9 for cross-classification is presented in this appendix.

Table G.1: Matlab® Functions for UAV Solution 9

subsystemsKBN09( )	Cross-classification from structures' point of view.
demo_subsys09( )	Plots runs of subsystemsKBN09, e.g. Fig. 8.11.
experiment09( )	Generates errors for subsystemsKBN09, e.g. Fig. 8.12, 8.13.

```
function [n, nn] = subsystemsKBN09(n,nn,M,nsys,nnaero,alpha)

   %Cross-classification with Aero
   %The search strategy is explore/exploit according to alpha.
   %The sigmas are rule-based.

   if isempty(n)
       n = kbn(2,1,2,'bnd',[1 .1;15 .4]);
       n.hscale = 0.4*ones(1,n.Din);
   end
   if isempty(nn)
       nn = kbn(2,1,2,'bnd',[1 .1;15 .4]);
       nn.hscale = ones(1,n.Din);
   end

   for i=1:M

       %get the next data point
       rn = rand(1,2);
       if n.N<10
           %exploratory
           xi = kbnSampleExplore(n,1);
       elseif n.Nc(1)>0 && rn(1)<alpha(1)
           %from the systems' acceptable design space
           xi = kbnSample(n,1,1);
       elseif nn.Nc(1)>0 && rn(2)<alpha(2)
           %from the systems' search domain
           xi = kbnSample(nn,1,1);
       else
           %exploratory
           xi = kbnSampleExplore(n,1);
       end
       yi = structF0((xi./n.scale+n.shift));
       %is the weight within the systems search domain?
       if yi>=nsys.bnd(1,2) && yi<=nsys.bnd(2,2)
           %is the design point within aero's acceptable drag domain?
           [caeroTest] = kbnEvalC(nnaero,xi);
           if caeroTest(1)==1
               %construct the aero kbn classifier
               naero = kbn(2,1,2,'bnd',[1 .1;15 .4]);
               naero.hscale = 0.4*ones(1,n.Din);
               %cross-classify aero's temporary classifier
               for j=1:nnaero.N
                   yitest = nsys.scale.*([nnaero.d(j,3) yi]-
   nsys.shift);
                   citest = kbnEvalC(nsys,yitest);
                   if citest(1)==0
                       citest(1) = 2;
                   end
                   naero = kbnAddData(naero,nnaero.d(j,:),citest(1));
               end
               naero.h = kbnEvalH(naero);
               %use it to classify struct's design point
               ci = kbnEvalC(naero,xi);
               n = kbnAddData(n,[xi yi],ci(1));
               nn = kbnAddData(nn,[xi yi],1);
           else
               n = kbnAddData(n,[xi yi],2);
               nn = kbnAddData(nn,[xi yi],2);
           end
       else
           n = kbnAddData(n,[xi yi],2);
           nn = kbnAddData(nn,[xi yi],2);
       end

       n.h = kbnEvalH(n);
       nn.h = kbnEvalH(nn);

   end

end


function demo_subsys09()

    %Cross-classification for structures
    %KBN exploratory sequence at the systems level
    %rule-based sigmas
    %fully connected BN

    I = 10; %repeat I times
    M = 10; %the number of data points per each of I iterations
    fhn = figure();
    ahn = axes();
    fhnn = figure();
    ahnn = axes();

    %information for the correct decision boundary
    fh = fopen('dataTestsolutionN10000.csv');
    [xsTestSol,ysTestSol] = dataRead(fh);
    fclose(fh);

    %get the systems bn
    nsys = systemsKBN11(struct([]),100);
    %get aero's bn
    nnaero = kbn(2,1,2,'bnd',[1 .1;15 .4]);
    nnaero.hscale = ones(1,nnaero.Din);
    xaero = halton(nnaero.Din,100+1);
    yaero = zeros(100,1);
    for i=1:100
        yaero(i,1) = aeroF0((xaero(i+1,:)./nnaero.scale+nnaero.shift));
        if yaero(i,1)>=nsys.bnd(1,1) && yaero(i,1)<=nsys.bnd(2,1)
            nnaero = kbnAddData(nnaero,[xaero(i+1,:) yaero(i,1)],1);
        else
            nnaero = kbnAddData(nnaero,[xaero(i+1,:) yaero(i,1)],2);
        end
    end
    nnaero.h = kbnEvalH(nnaero);

    n = struct([]); %start with the "empty" bn
    nn = struct([]);
    for i=1:I
        [n nn] = subsystemsKBN09(n,nn,M,nsys,nnaero,[0 0]);

        figure(fhn)
        cla(ahn);
        Pc1 = (n.Nc(1)+1)/(n.N+2);
        Pc2 = (n.Nc(2)+1)/(n.N+2);
        [xh1 ph1 dph1] = kbnPlot(n,[.025 .025],[n.lf(1)*Pc1
n.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),n.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'EdgeC
olor','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),n.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'EdgeC
olor','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:n.N
            pTemp = kbnEval(n,n.d(j,1:n.Din));
            pDiff = (n.lf(1)*Pc1*pTemp(1,1,1)-
n.lf(2)*Pc2*pTemp(1,1,2));
            if n.w(j,1)>0
                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),'b>','MarkerFaceColor','b','MarkerSize',8); hold
on;
                end
            else
                if pDiff>0

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                else

plot(n.d(j,1),n.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8); hold
on;
                end
            end
        end
        %plot the decision surface

%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);

        figure(fhnn)
        cla(ahnn);
        Pc1 = (nn.Nc(1)+1)/(nn.N+2);
        Pc2 = (nn.Nc(2)+1)/(nn.N+2);
        [xh1 ph1 dph1] = kbnPlot(nn,[.025 .025],[nn.lf(1)*Pc1
nn.lf(2)*Pc2]);
        %plot the kbn surfaces

surface(xh1(:,1),xh1(:,2),nn.lf(1)*Pc1*ph1(:,:,1),'FaceAlpha',0.2,'Edge
Color','b','FaceColor','b'); hold on;

surface(xh1(:,1),xh1(:,2),nn.lf(2)*Pc2*ph1(:,:,2),'FaceAlpha',0.2,'Edge
Color','r','FaceColor','r'); hold on;
        %plot the data points
        for j=1:nn.N
            pTemp = kbnEval(nn,nn.d(j,1:nn.Din));
            pDiff = (nn.lf(1)*Pc1*pTemp(1,1,1)-
nn.lf(2)*Pc2*pTemp(1,1,2));
            if nn.w(j,1)>0
                if pDiff>0

plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
```

```
                else
                    
plot(nn.d(j,1),nn.d(j,2),'b>','MarkerFaceColor','b','MarkerSize',8);
hold on;
                end
            else
                if pDiff>0
                    
plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
                else
                    
plot(nn.d(j,1),nn.d(j,2),'r<','MarkerFaceColor','r','MarkerSize',8);
hold on;
                end
            end
        end
        %plot the decision surface
        
%surface(xh1(:,1),xh1(:,2),dph1,'FaceAlpha',0.2,'EdgeColor','g','FaceCo
lor','g'); hold on;
        %plot the decision boundary
        contour(xh1(:,1),xh1(:,2),dph1,[0 0],'k--','LineWidth',2); hold
on;
        axis([0 1 0 1]);
        
        pause();
    end
    
    %plot the correct decision boundary
    figure(fhn);
    axis(ahn);
    tocTestSol = zeros(1,100);
    chordTestSol = zeros(1,100);
    rangeTestSol = zeros(100,100);
    dragTestSol = zeros(100,100);
    weightTestSol = zeros(100,100);
    for i=1:100
        tocTestSol(i) = n.scale(1)*(xsTestSol(i,1)-n.shift(1));
        chordTestSol(i) = n.scale(2)*(xsTestSol((i-1)*100+1,2)-
n.shift(2));
        rangeTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,3);
        dragTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,1);
        weightTestSol(i,:) = ysTestSol((i-1)*100+1:i*100,2);
    end
    contour(tocTestSol,chordTestSol,rangeTestSol,[900
900],'k','LineWidth',2); hold on;
    
    figure(fhnn);
    axis(ahnn);
    contour(tocTestSol,chordTestSol,dragTestSol,[1
1],'k','LineWidth',2); hold on;
```

```
    contour(tocTestSol,chordTestSol,weightTestSol,[5
5],'k','LineWidth',2); hold on;

end


function experiment09()

    %Cross-classification for structures
    %Systems and Aero:
    %  halton sequence
    %  rule-based sigmas: new rule
    %  parzen window

    N = 100;

    %error test points for subsystems
    fh = fopen('dataTestsolutionN1000.csv');
    [xTestSS,yTestSS,~,~,NtotTestSS] = dataRead(fh);
    cTestSS = 2*ones(NtotTestSS,1);
    cTestSS(yTestSS(:,3)>=900) = 1;
    fclose(fh);

    %error output files
    fhss1 = fopen('exp09_sse1.csv','w'); %false positives for
subsystems
    fhss2 = fopen('exp09_sse2.csv','w'); %false negatives for
subsystems

    %get the systems bn
    nsys = systemsKBN11(struct([]),100);
    %get aero's bn
    nnaero = kbn(2,1,2,'bnd',[1 .1;15 .4]);
    nnaero.hscale = ones(1,nnaero.Din);
    xaero = halton(nnaero.Din,100+1);
    yaero = zeros(100,1);
    for i=1:100
        yaero(i,1) = aeroF0((xaero(i+1,:)./nnaero.scale+nnaero.shift));
        if yaero(i,1)>=nsys.bnd(1,1) && yaero(i,1)<=nsys.bnd(2,1)
            nnaero = kbnAddData(nnaero,[xaero(i+1,:) yaero(i,1)],1);
        else
            nnaero = kbnAddData(nnaero,[xaero(i+1,:) yaero(i,1)],2);
        end
    end
    nnaero.h = kbnEvalH(nnaero);

    its = 1;
    while its<=300

        nssys = struct([]);
        nnssys = struct([]);
        %subsystems classification
        for i=1:N
            [nssys nnssys] =
subsystemsKBN09(nssys,nnssys,1,nsys,nnaero,[0 0]);
            [e1 e2] = kbnGetErr(nssys,xTestSS,[1 2],cTestSS);
            fprintf(fhss1,'%g,',e1);
            fprintf(fhss2,'%g,',e2);
        end
        fprintf(fhss1,'\n');
        fprintf(fhss2,'\n');

        its = its+1;
    end
    fclose(fhss1);
    fclose(fhss2);

end
```

Bibliography

Abott, I. H. and A. E. von Doenhoff, 1959, *Theory of Wing Sections: Including a Summary of Airfoil Data*, Dover Publications, Inc., New York.

Allison, J., M. Kokkolaras, M. Zawislak and P. Y. Papalambros, 2005, "On the Use of Analytical Target Cascading and Collaborative Optimization for Complex System Design", *6th World Congress on Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brazil.

Allison, J. T., M. Kokkolaras and P. Y. Papalambros, 2007, "Optimal Partitioning and Coordination Decisions in Decomposition-Based Design Optimization", *ASME IDETC/CIE*, Las Vegas, NV, Paper No. DETC2007-34698.

Batill, S. M., M. A. Stelmak, and R. S. Sellar, 1999, "Framework for Multidisciplinary Design Based on Response-Surface Approximations", *Journal of Aircraft*, Vol. 36, No. 1, pp. 287-297.

Bosman, P. A. N. and D. Thierens, 2000, "ID𝔼As Based On the Normal Kernels Probability Density Function", *Utrecht University Technical Report* UU-CS-2000-15.

Bowman, A. W. and A. Azzalini, 1997, *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*, Oxford University Press Inc., New York.

Browning, T. R., and S. D. Eppinger, 2002, "Modeling Impacts of Process Architecture on Cost and Schedule Risk in Product Development", *IEEE Transactions on Engineering Management*, **49**(4), pp. 428-442.

Carlos, S., K. Madhavan, G. Gupta, D. A. Keese, U. Maheshwaraa and C. C. Seepersad, 2006, "A Flexibility-Based Approach to Collaboration in Multiscale Design," *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Portsmouth, VA.

Carlson, D., M. Malone, J. Kollat, and T. W. Simpson, 2008, "Evaluating the Performance of Visual Steering Commands for User-Guided Pareto Frontier Sampling During Trade Space Exploration", *ASME DETC/CIE*, Brooklyn, NY, Paper No. DETC2008/DAC-49681.

Cerny, V., 1985, "A Thermodynamic Approach to the Traveling Salesperson Problem: An Efficient Simulation Algorithm", *Journal of Optimization Theory and Applications*, Vol. 45, No. 1, pp. 41-51.

Chang, T.-S., Ward, A. C., 1995, "Conceptual Robustness in Simultaneous Engineering: A Formulation in Continuous Spaces", *Research in Engineering Design*, Vol. 7, pp. 67-85.

Chen, W. and K. Lewis, 1999, "A Robust Design Approach for Achieving Flexibility in Multidisciplinary Design," *AIAA Journal,* Vol. 37, No. 8, pp. 982-989.

Cook, R. D. and W. C. Young, 1999, Advanced Mechanics of Materials, 2nd ed., Prentice Hall, Inc., Upper Saddle River, NJ.

Davies, S. and A. Moore, 2000, "Mix nets: Factored Mixtures of Gaussians in Bayesian Networks with Mixed Continuous and Discrete Variables", *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, (UAI2000).*

Dudda, R. O., P. E. Hart and D. G. Stork, 2001, *Pattern Classification*, 2nd Ed., John Wiley & Sons, Inc., New York.

Duivesteijn, W. and A. Feelders, 2008, "Nearest Neighbour Classification with Monotonicity Constraints", *Machine Learning and Knowledge Discovery in Databases*, W. Daelemans, B. Goethals, K. Morik, editors, Springer-Verlag, Berlin.

Eiben, A. E. and C. A. Schippers, "On Evolutionary exploration and exploitation", *Fundamenta Informaticae*, Vol. 35, pp. 1-16.

Finch, W. W. and A. C. Ward, 1996, "Quantified Relations: A Class of Predicate Logic Design Constraints Among Sets of Manufacturing, Operating, and other Variations", *ASME DETC/CIE*, Paper No. 96-DETC/DTM-1278, Irvine, CA, August 18-22.

Finch, W. W. and A. C. Ward, 1997, "A Set-Based System for Eliminating Infeasible Designs in Engineering Problems Dominated by Uncertainty", *ASME DETC/CIE*, Paper No. DETC97/DTM-3386, Sacramento, CA, September 14-17.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wessley, Reading, MA.

Gunawan, S., A. Farhang-Mehr and S. Azarm, 2004, "On Maximizing Solution Diversity in a Multiobjective Multidisciplinary Genetic Algorithm for Design Optimization," *Mechanics Based Design of Structures and Machines*, Vol. 32, No. 4, pp. 491-514.

Halton, J. H., 1960, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals", *Numerische Mathematik*, Vol. 2, pp. 84-90.

Hammersley, J. M., 1960, "Monte Carlo Methods for Solving Multivariable Problems", *Annals of the New York Academy of Sciences*, Vol. 86, pp. 844-874.

Hanna, L. and J. Cagan, 2010, "Protocal-based Multi-Agent Systems: Examining the Effect of Diversity, Dynamism, and Cooperation in Heuristic Optimization Approaches", *ASME IDETC/CIE*, Montreal, Quebec, Canada, Paper No. DETC2010-28601.

Hoerner, S. F., 1965, *Fluid Dynamic Drag*, published by the author, Midland Park, NJ.

Hoffman, R. and V. Tresp, 1996, "Discovering Structure in Continuous Variables Using Bayesian Networks", *Advanvces in Neural Information Processing Systems 8*, D. S. Touretzsky, M. C. Mozer and M. Hasselmo, editors, MIT Press, Cambridge MA.

Holland, J., 1992, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Control, and Artificial Intelligence*, MIT Press, Cambridge, MA.

Ivezic, N. and J. H. Garret, 1998, "Machine Learning for Simulation-Based Support of Early Collaborative Design", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 12, pp. 123-139.

Jensen, F. V and T. D. Nielsen, 2007, *Bayesian Networks and Decision Graphs*, Springer Science + Business Media, LLC, New York, NY.

Jin, R., W. Chen and A. Sudjianto, 2002, "On Sequential Sampling for Global Metamodeling in Engineering Design", *ASME IDETC/CIE*, Montreal, Canada, Paper No. DETC2002-34092.

John, G. H. and P. Langley, 1995, "Estimating Continuous Distributions in Bayesian Classifiers", *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence,* Morgan Kaufmann Publishers, San Mateo.

Johnson, M. E., L. M. Moore and D. Ylvisaker, 1990, "Minimax and Maximin distance designs", *Journal of Statistical Planning and Inference*, Vol. 26, Iss. 2, pp. 131-148.

Jones, D. R., 2001, "A Taxonomy of Global Optimization Methods Based on Response Surfaces", *Journal of Global Optimization*, Vol. 21, pp. 345-383.

Jourdan, A., and J. Franco, 2009, "A New Criterion Based on Kullback-Leibler information for space filling designs", *ArXiv e-prints*.

Katz, J., and A. Plotkin, 2002, *Low Speed Aerodynamics*, Cambridge University Press, Cambridge.

Kennedy, J., and R. C. Eberhart, 1995, "Particle Swarm Optimization" *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, Nov. 1995, pp. 1942-1948.

Kikpatrick, S., C. D. Gelatt and M. P. Vecchi, 1983, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4958, pp. 671-680.

Kim, H. M., N. F. Michelena, P. Y. Papalambros, and T. Jiang, 2003, "Target Cascading in Optimal System Design," *ASME Journal of Mechanical Design*, Vol. 125, pp. 474-480.

Koehler, J. R. and A. B. Owen, 1996, "Computer Experiments", *Handbook of Statistics, Volume 13*, S. Ghosh and C. R. Rao editors., Elsevier Science, New York, pp. 261-308.

Kroo, I., S. Altus, R. Braun, P. Gage, and J. Sobieski, 1994, "Multidisciplinary Optimization Methods for Aircraft Preliminary Design," 5th *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-94-4325, Panama City, FL, **1**, pp. 697-707.

Kuncheva, L. I., 2004, *Combining Classifiers: Methods and Algorithms*, John Wiley and Sons, Inc., Hoboken, NJ.

Lennon, A., 1996, *Basics of R/C Model Aircraft Design*, Air Age Media Inc., Ridgefield, CT.

Lewis, K. and F. Mistree, 1998, "Collaborative, Sequential and Isolated Decisions in Design," *ASME Journal of Mechanical Design*, Vol. 120, No. 4, pp. 643-652.

Liao, Y., S. Fang and H. L. W. Nuttle, 2003, "Relaxed conditions for radial-bases function networks to be universal approximators", *Neural Networks*, Vol. 16, Issue 7, pp. 1019-1028.

Liu, H., W. Chen, and M. J. Scott, 2008, "Determination of Ranged Sets of Design Specifications by Incorporating Heterogeneous Design Space Heterogeneity", *Engineering Optimization*, Vol. 40, Iss. 11, pp. 1011-1029.

Lottaz, C., I. F. C. Smith, Y. Robert-Nicoud, and B. V. Faltings, 2000, “Constraint-Based Support for Negotiation in Collaborative Design”, *Artificial Intelligence in Engineering*, Vol. 14, pp. 261-280.

Madhavan, K., 2007, "A Framework for a Flexibility-Based Approach to Multiscale and Multidisciplinary Design,” *M.S. Thesis*, Mechanical Engineering Department, The University of Texas at Austin, Austin, TX.

Madhavan, K., D. Shahan, C. C. Seepersad, D. A. Hlavinka, W. Benson, 2008, “An Industrial Trial of a Set-Based Approach to Collaborative Design”*, ASME DETC/CIE*, Brooklyn, NY, Paper Number DETC2008/49953.

Malak, R. J. and C. J. J. Paredis, 2007, “Using Parameterized Pareto Sets to Model Design Concepts”, *ASME IMECE*, Seattle, WA, Paper No. IMECE2007-43226.

Malak, R. J., J. M. Aughenbaugh, C. J. J. Paredis, 2009, “Multi-attribute Utility Analysis in Set-Based Conceptual Design”, *Computer-Aided Design*, Vol. 41, Issue 3, pp. 214-227.

Mckay, M. D., R. J. Beckman and W. J. Conover, 1979, “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”, *Technometrics*, Vol. 21, pp. 239-245.

Michelena, N., H. Park and P. Y. Papalambros, 2003, “Convergence Properties of Analytical Target Cascading”, *AIAA Journal*, Vol. 41, No. 5.

Mistree, F., O. F. Hughes and B. A. Bras, 1993, "The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm," *Structural Optimization: Status and Promise*, AIAA, Washington, D.C., pp. 247-286.

Moore, R. E., 1966, *Interval Analysis*, Prentice-Hall, New York.

Moran, J., 1984, *An Introduction to Theoretical and Computational Aerodynamics*, John Wiley & Sons, New York.

Nair, P. B. and A. J. Keane, 2002, "Coevolutionary Architecture for Distributed Optimization of Complex Coupled Systems," *AIAA Journal*, Vol. 40, No. 7, pp. 1434-1443.

Otto, K. N. and K. E. Antonsson, 1991, "Trade-off Strategies in Engineering Design", *Research in Engineering Design*, Vol. 3, No. 2, pp. 87-104.

Otto, K. N. and K. L. Wood, 1995, "Estimating Errors in Concept Selection", *Design Engineering Technical Conferences*, Vol. 2, pp. 397-411.

Panchal, J. H., M. G. Fernandez, C. J. J. Paredis, J. K. Allen, and F. Mistree, 2007, "Interval-based Constraint Satisfaction (IBCS) Method for Decentralized, Collaborative Multifunctional Design", *Concurrent Engineering: Research and Applications*, Vol. 15, No. 3, pp. 309-323.

Parzen, E., 1962, "On Estimation of Probability Density Function and Mode", *Annals of Mathematical Statistics*, Vol. 33, pp. 1065-1076.

Pearl, J., 1988, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 2nd Ed., Morgan Kauffman Publishers, Inc., San Francisco.

Perez, A., P. Larranga and I. Inza, 2009, "Bayesian classifiers based on kernel density estimation: Flexible classifiers", *International Journal of Approximate Reasoning*, 50, pp. 341-362.

Powell, M. J. D., 1987, "Radial basis functions for multivariate interpolation: A review", *Algorithms for Approximation*, J. C. Mason and M. G. Cox editors, Oxford, pp. 143-167.

Rai, R. and M. Campbell, 2008, "Q2S2: A New Methodology for Merging Quantitative and Qualitative Information in Experimental Design", *Journal of Mechanical Design*, Vol. 130.

Raymer, D. P., 2006, *Aircraft Design: A Conceptual Approach*, 4th ed., AIAA, Inc., Reston, VA.

Roth, B., and M. Kroo, 2008, “Enhanced Collaborative Optimization: A Decomposition-Based Method for Multidisciplinary Design”, *ASME DETC/CIE*, Paper Number: DETC2008-50038, Brooklyn, NY, August 3-6.

Sasena, M., 2002, *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

Scott, D. W., 1992, *Multivariate Density Estimation*, John Wiley & Sons, Inc., New York.

Scott, M. J. and E. K. Antonsson, 1996, “Formalisms for Negotiation in Engineering Design”, *ASME-DETC/CIE*, Paper Number: 96-DETC/DTM-1525, Irvine, CA, August 18-22.

Seepersad, C. C., D. Shahan, K. Madhavan, 2007, “Multiscale Design for Solid Freeform Fabrication”, *Solid Freeform Fabrication Symposium*, Austin, TX, USA.

Silverman, B. W., 1986, *Density Estimation*, Chapman and Hall, London.

Sheather, S. J., 2004, “Density Estimation”, *Statistical Science*, Vol. 19, No. 4, pp. 588-597.

Shahan, D., C. C. Seepersad, 2010, “The Implications of Alternative Multiscale Design Methods for Design Process Management”, *Concurrent Engineering: Research and Applications*, Vol. 18, No. 1, pp. 5-18.

Siminoff, J. S., 1996, *Smoothing Methods in Statistics*, Springer-Verlag, New York, NY.

Sobek, D. K., A. Ward and J. K. Liker, 1999, “Toyota's Principles of Set-Based Concurrent Engineering,” *Sloan Management Review*, Winter 1999, pp. 67-83.

Sobieski, J., 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*, NASA TM-101494, NASA CP-3031, Hampton, VA.

Sobieszczanski-Sobieski, J., 1992, "Aircraft Optimization by a System Approach: Achievments and Trends", *NASA Technical Memorandum*, NASA TM-107662.

Sobieszczanski-Sobieski, J. and R. J. Balling, 1994, "Optimization of Coupled Systems: a Critical Overview of Approaches", *ICASE Report,* Report No. 94-100.

Sobieszczanski-Sobieski, J. and R. T. Haftka, 1997, "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments", *Structural Optimization*, Vol. 14, pp. 1-23.

Sobieski, J., J. S. Agte, and R. R. Sandusky, 2000, "Bilevel Integrated System Synthesis", *AIAA Journal*, Vol. 38, No. 1. pp. 164-172.

Sohn, E., 2003, "Model Plane Flies the Atlantic", http://www.sciencenewsforkids.org/articles/20031217/Feature1.asp.

Steward, D. V., 1981, *Systems Analysis and Management: Structure, Strategy, and Design*, Petrocelli Books, New York.

Tipping, M. E., 2001, "Sparse Bayesian Learning and the Relevance Vector Machine", *Journal of Machine Learning Research*, 1 (2001), pp. 211-244

Turner, C. J., R. H. Crawford, M. I. Campbell, 2007, "Multidimensional Sequential Sampling for NURBs-Based Metamodel Development", *Engineering with Computers*, Vol. 23, pp. 155-174.

U.S. Standard Atmosphere, 1976, U.S. Government Printing Office, Washinton, D. C.

Vapnik, V., S. Golowich, and A. Smola, 1997, "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing", *Advances in Neural Information Processing Systems,* Vol. 9, pp. 281-287.

Venter, G., and J. Sobieszczanski-Sobieski, 2004, "Multidisciplinary Optimization of a Transport Wing Using Particle Swarm Optimization", *Structural and Multidisciplinary Optimizaation*, Vol. 26, pp. 121-131.

Ward, A., 1989, *A Theory of Quantitative Inference Applied to a Mechanical Design Compiler*, PhD Thesis, MIT, Cambridge.

Ward, A., J. K. Liker, J. J. Cristiano, and D. K. Sobek, 1995, "The Second Toyota Paradox: How Delaying Decisions Can Make Cars Faster", *Sloan Management Review*, Spring 1995, pp. 43-61.

Web, A., 2002, *Statistical Pattern Recognition*, 2nd Ed., John Wiley & Sons, Ltd., West Sussex, England.

Wood, K. L., and E. K. Antonsson, 1988, "Computations with Imprecise Parameters in Engineering Design: Background and Theory", *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 111, No. 4, pp. 616-625.

Wood, K. L., E. K. Antonsson,and J. L. Beck, 1990, "Representing Imprecision in Engineering Design: Comparing Fuzzy and Probability Calculus", *Research in Engineering Design*, Vol 1., pp. 187-203.

Xiang, Y., 1994, "Probabilistic Framework for Multi-Agent Distributed Interpretation and Optimization of Communication", *Artificial Intelligence*, Vol. 87, Iss. 1-2, pp. 295-342.

Xiang, Y. and V. Lesser, 2003, "On the Role of Multiply Sectioned Bayesian Networks to Cooperative Multiagent Systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans,* Vol. 33, Iss. 4, pp. 489-501.

Xiang, Y., J. Chen, and A. Deshmukh, 2004, "A Decision-Theoretic Graphical Model for Collaborative Design on Supply Chains", *Advances in Artificial Intelligence*, LNAI 3060, pp. 355-369.

Vita

David Shahan received his BS in Mechanical Engineering at Columbia University's School of Engineering and Applied Science in 1997. He worked at Pratt & Whitney from 1998 to 2006 as a senior mechanical engineer, studying at night to receive his MS in Mechanical Engineering at Rensselaer at Hartford as well as mentor Hartford Public High School students for the US FIRST robotics competition. He has continued his graduate studies at the University of Texas at Austin, pursuing a PhD in Mechanical Engineering since 2006.

Permanent address and email:
80 Red River St. Apt. 320
Austin, TX 78701
david.shahan@gmail.com

This dissertation was typed by David Shahan.