A PRODUCT FAMILY DESIGN METHODOLOGY EMPLOYING PATTERN RECOGNITION

A Thesis
Presented to
The Academic Faculty

by

Dane F. Freeman

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
December 2013

A PRODUCT FAMILY DESIGN METHODOLOGY EMPLOYING PATTERN RECOGNITION

Approved by:

Prof. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Gunnar Holmberg
Saab Aerosystems
Saab AB

Dr. Dongwook Lim
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 21 August 2013

To my parents

ACKNOWLEDGEMENTS

The Ph.D. process is long and arduous and I owe much to those who have helped me along the way. I want to begin with thanking my committee members Dr. Mavris, Dr. Holmberg, Dr. German, Dr. Schrage, and Dr. Lim for their challenging questions and feedback to help produce the final product.

I would like to thank Dr. Mavris for giving me the opportunity to grow academically at the Aerospace Systems Design Laboratory. ASDL is a unique environment with many projects and people to learn from. I also want to call special attention to Dr. Holmberg and Saab's support for the past few years as I have worked towards completing this dissertation. Additionally, I want to thank Dr. Lim and Dr. Garcia for their guidance and supervision of my research over the years. I would also like to thank Philippe Ranque for his contributions to the aircraft modeling and simulation environment.

I also want to acknowledge a few of my colleagues at ASDL: Curtis Iwata, James Arruda, Evan Anzalone, Carl Johnson, Elizabeth Tang, Matt Daskilewicz, Jonathan Murphy, and Chung Lee. I appreciate their time and effort listening to me sounding out my research ideas and found their insight irreplaceable. My time in graduate school was much more meaningful through their friendship and support.

Most importantly, I would like to acknowledge my loving family, my parents who have helped instill in me a life long love of learning, and my sister Danielle and her husband Michael for being there when I needed them.

Finally, with my document complete, I look forward to enjoying everyone's company again.

Thank you.

Contents

List of Tables

List of Figures

NOMENCLATURE

$\bar{x}$	Vector of x
$\mathbb{E}(x)$	Expected Value of a Random Variable
$\mathscr{N}(\mu, \sigma)$	Normal Distribution for a Random Variable
$\mathcal{P}$	Set of Products in a Family
μ	Mean
$\phi(x, x')$	Kernel Basis Function
Σ	Covariance Matrix
σ	Standard Deviation
b	Number of Basis Functions
$P(x, y)$	Joint Distribution of x, y
$P(x)$	Probability of a Random Variable x
R_{ij}^{pq}	Binary relation variable indicating sharing design variables between products p and q
XB	Xie Beni Index
BIC	Bayesian Information Criterion
CDE	Conditional Density Estimation
CDF	Cumulative distribution function
Component	A piece of a product that may be indivisible or composed of two or more interrelated sub-components

DAG	Directed Acyclic Graph
DoE	Design of Experiments
EMD	Earth Movers Distance
FCM	Fuzzy C-Means
LHC	Latin Hypercube
LSCDE	Least Squares Conditional Density Estimation
MC	Monte Carlo
Module	A physical or conceptual grouping of components that share some characteristics
MoE	Measure of Effectiveness
MOGA	Multiobjective Genetic Algorithm
PDF	Probability density function
PMBGA	Probabilistic Model Building Genetic Algorithm
Product Family	A set of similar products that are derived from common platforms and yet still possess specific features or functionality to meet particular customer requirements
Product Platform	A set of subsystems and interfaces developed to form a common structure from which a stream of derivative products can be efficiently developed and produced
SAR	Search and Rescue
SoS	System-of-systems
UAVs	Unmanned Air Vehicles

SUMMARY

To remain competitive in the marketplace, companies must offer their products at competitive prices. If they create many products to fulfill similar but different market niches, then there are implied similarities between these products. The company can then attempt to leverage those similarities and create a family of products to streamline design, improve manufacturing, and facilitate maintenance.

Sharing components, called platforms, between different products can minimize duplication of effort, thereby lowering family costs. Additional cost savings comes through economies of scale as the same component has now multiple end products. However, if the products' requirements are too dissimilar, sharing components may compromise the end product; such variance will lead to lower end products being overdesigned and/or higher end products being underdesigned. It is critical for a successful family to identify which components are similar, so that sharing does not compromise the individual products' performances.

Most existing product family design methods make decisions a priori about platforms; restricting platforms to be used by every product in the family. Methods that simultaneously optimize component sharing and design variable settings have the potential to find better families. However, allowing components to be shared between any subset of products leads to a very large combinatorial problem. Considering large product families can be difficult to explore because of high computationally complexity.

In addition to the combinatorial problem, the computational complexity is also driven by the scope of the family and their operating environment. Existing methods focus on families with a set of well defined requirements. However, these requirements are not known at the conceptual design phase for many problems. This is especially true when the products in the family operate cooperatively to achieve mission objectives. Products operating in this system-of-systems manner require more complex analysis models which have longer execution times. System-of-systems problems are also usually stochastic which requires again increasing the number of evaluations and model complexity. A system-of-systems product family with unknown component sharing quickly becomes intractable.

A new generic product family design methodology is proposed, which attempts to lessen the combinatorial problems by identifying possible sets of commonality inherent to the family. By having a commonality approach that can reduce the combinatorial problem allows for more family alternatives to be considered. The methodology needs to balance cost savings and performance compromises due to component commonality and should address product growth potential, technology evolution, and uncertainty. Also, it should be possible to determine the platform sensitivity to changing requirements. This research formulates and tests two commonality identification approaches.

The first, a clustering approach, is based on the pattern recognition technique of fuzzy c-means clustering in domain subspaces. While these domains exist across the entire scope of product family design, this dissertation focuses on component subspaces. If components from different products are similar enough to be grouped into the same cluster, then those components could possibly become the same platform. Fuzzy equivalence relations that show the binary relationship from one products' component to a different products' component can be extracted from the cluster membership functions.

The second, a probabilistic approach, focuses on treating the results of a design space exploration as a joint probability distribution. This joint probability distribution is then encoded by a continuous nonparametric Bayesian network. The Bayesian network acts as a surrogate model of the design space. However, unlike traditional surrogate models the network captures more information about the interactions between the design variables. Additionally, the Bayesian network provides a robust evaluation framework of statistical inference using importance sampling. Through inferences, posterior distributions of the design variables can be obtained by conditioning on a set of product performances and component constraints. Finally, the posterior design variable distributions are processed using a similarity metric like the earth mover distance to identify which products' components are similar to another's.

To understand the applicability and limits of these commonality identification approaches a series of experiments and a practical demonstration problem are presented. The experiments consist of a family design problem of universal electric motors. From these experiments, the fuzzy clustering was found to be a sound approach to extract commonality. However the approach scales poorly with large complicated designs requiring complex models with long execution times. These experiments found the probabilistic approach to be more accurate and scalable to large complex product design spaces.

Finally, a practical aircraft family design problem is presented demonstrating the generic product family design methodology and both commonality identification approaches. These aircraft operate together to perform a variety of missions across two unique scenarios. This requires the implementation of a system-of-systems model for each scenario. The first scenario is a set of maritime monitoring missions in the Norwegian Exclusive Economic Zone (EEZ). The second scenario is an aerial firefighting mission where the aircraft find and extinguish the fire. The goal is to determine whether the commonality approaches can detect changes in component similarities using different design criteria.

The design of product families requires solving a large combinatorial problem. An approach to quantify component similarities serves to decrease this combinatorial problem by pruning poor options from consideration and provides valuable feedback for designers. Although the fuzzy clustering approach can be limited to small problems, probabilistic approach using Bayesian networks is a robust technique for assessing commonality. Furthermore, the use of Bayesian networks as a source for encoding the joint probability distribution of a design space has implications well beyond the study of product families.

Chapter I

PRODUCT FAMILY OVERVIEW

Customers desire low cost products that satisfy their needs. For a manufacturer to capture a significant market share, it must offer a variety of products each satisfying targeted customer niches in a timely and inexpensive manner. To accomplish this goal, manufacturers can either design each product independently or as a group. If several targeted customer market niches are similar enough, then there could be some commonality between different products and they could share common components. Groups of similar products that share common components are called product families. Sharing components between different products can minimize duplication of effort, thereby lowering family costs. Efficiency in design, through careful development of product families, increases revenue for both manufacturer and customer.

1.1 Product Families

This thesis analyzes the relationship of product families and individual products with respect to shared components. A product, in this instance, is defined as a component which is offered to a customer[150]. In general, a product may be a service, a process, or anything which is offered to the market. This thesis, however, is concerned with physical products, rather than processes, that are comprised of physical components. A component refers to an object and may be indivisible or composed of two or more interrelated sub-components. The distinction between products and components is fuzzy and depends on perspective, because what is sold as a product by one supplier may become a component of a subsystem of another company.

A product family is a set of similar products that are derived from common platforms but possess specific features or functionality to meet particular customer requirements [101]. The concept of a product family, however, is not limited to only the physically existing products. This concept can also encompass other variants, such as possible future products[150].

Individual products in a family are thus defined by their composition of shared components (product platforms) and unique components. A product platform is defined as a set of subsystems and interfaces developed to form a common structure from which a stream of derivative products can be efficiently developed and produced [101]. It is the common set of design variables around which a family of products can be developed [150]. The idea of platforms also extends to manufacturing technologies and processes employed in production [101].

Product platform design theory came from the mass customization paradigm of the late 1980s [31]. Mass customization refers to the ability to provide products, tailored to the needs of multiple customer niches, by reusing components common across a family of products[35]. Firms who manage a product portfolio one single product at a time fail to embrace commonality, compatibility, and standardization[101]. This failure to embrace commonality misses an opportunity for a better performing family because it leads to the development of several unique but similar product specific sub-components [101]. The single product focus is common with evolutionary product approaches. One example of a manufacturer rectifying their evolutionary family is in the 1970s when Black & Decker effectively redesigned their hand held tools by treating the family as a whole[101]. As a result, the newly designed set of products were based on similar components which lowered costs.

1.1.1 Advantages of Product Families

There are several benefits of product platforms. The primary benefit is lower life cycle costs. The majority of the cost savings associated with product platforms is derived from the manufacturer's ability to reduce redundant effort and waste. Moreover, because components are used between products, there is a greater capitalization on economies of scale [57]. Economies of scale, in turn, increase cost-efficiency by reducing the fixed cost per unit as the volume of production increases. Even if the whole products themselves do not lead to economies of scale because of the need to maintain product variety, it is believed that at least the shared elements could enable similar economies of scale effects.

Product platforms also reduce maintenance costs and logistical problems because of a simplified supply chain. Because of the commonality between products fewer unique replacement components need to be stocked. Also maintenance personnel are more versatile and efficient as training can be applied to all of the similar products. Personnel can better maintain products because they are very familiar with normal behavior expected from wear and tear, and can replace or maintain components based on a set of reliable tested criteria.

In addition to lower life cycle costs, product families can be more reliable because component designers can focus more attention on components shared across the platform. Specifically, product reliability is further bolstered because fewer components allow design analysis to focus in more detail on how the part will be used and anticipate possible failure modes.

Platforms can also be leveraged in future products, making them vital to future core capabilities [100]. For instance, an aircraft manufacturer can add a fuselage plug increasing the interior volume for the next generation of aircraft. Thus, development time is also shortened on derivative products because manufacturers can use existing components should another product be desired and is within the capability range of the current family.

Another added benefit is the increased ability to upgrade an end product because a product platform can be easily modified through the addition, substitution, and exclusion of modules [147]. A module refers to a physical or conceptual grouping of components that share some characteristics [161]. Components that all operate the same way can easily be upgraded and improved upon. This adds flexibility, which is the ability of a design to satisfy changing requirements that occur after the system has been fielded[132]. Flexibility also extends product life by providing more options for upgradeability[163].

1.1.2 Disadvantages of Product Families

Product platforms have a few limitations and may not always offer the best means to reach the market [88]. The fixed development costs for a product platform can be as much as 10 times greater compared to single product development, primarily because of added complexity associated with coupling product platforms with the interfaces necessary between components[162].

Costs of platform-based product development inhibit its use for high levels of non-platform scale economies, or extreme levels of market diversity [88]. If the manufacturer will produce a large number of products which already enable economies of scale, then the added investment in platforms may not yield much of a portfolio improvement.

Extreme levels of market diversity means there is a large gap in requirements between the highest end products to the lowest end. The excess capability from a component designed for the constraining product may lead to a much more expensive lower end product than if it had been a unique component [47]. Increases to lower performance products' cost limit the amount of ideal commonality in product families. For example, a particular product's performance may degrade because of component sharing. Consider the design of an automobile family that employes a common engine across a high performance sports car and a lower performing fuel efficient car. Sharing the engine requires a trade study because the vehicles' performances are drastically opposing each other and the final performance is closely coupled to the engine. If the shared engine was designed primarily for use in the sports car, the fuel efficient car would have more excess power but also suffer from poorer fuel economy. If the engine was designed for the fuel efficient car and used in the sports car, the sports car would probably suffer a decrease in performance. Any engine designed to meet the middle of the road must trade-off fuel efficiency and excess power, but the end products may not satisfy the consumer market niches. Therefore, when determining the extent of platform development, care must be taken to maximize the benefit to the family while sacrificing the least performance.

This merging of product performances can also reduce customer perception of product differentiation. Distinctiveness is a difficult quality to predict and relates to consumer perception of the product. Figure 1 shows the trade-off between distinctiveness versus commonality for different architecture characteristics. Products that have no common parts are very distinctive. However, as sharing increases, products become less distinctive. The loss of distinctiveness is different for various product architectures. When deciding on a product architecture, some alternatives are less sensitive to this loss and may allow for better component commonality without sacrificing too much distinctiveness.

Eventually, no matter the architecture, as commonality increases, product performances within the family become more similar and may decrease compared to the highly optimized independent product with no commonality, e.g. the car example.

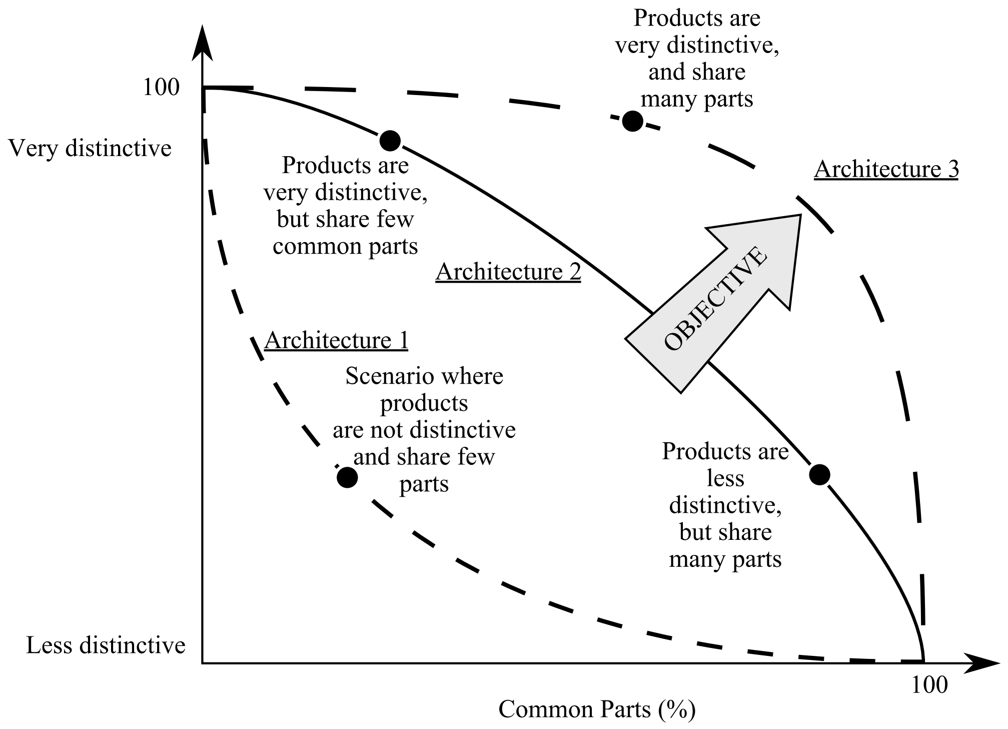

Products are very distinctive, and share many parts
100
Very distinctive
Products are very distinctive, but share few common parts
Architecture 3
Architecture 2
OBJECTIVE
Architecture 1
Scenario where products are not distinctive and share few parts
Products are less distinctive, but share many parts
Less distinctive
100
Common Parts (%)

Figure 1: Product Family Distinctiveness Trade-off[128]

1.1.3 Aerospace Examples of Product Families

There are several examples of product families in the aerospace industry. In aerospace, the products being designed are inherently complex due to highly coupled components necessary for meeting functional requirements. There is also high risk with large costs associated with their development. With current complex systems the trend is to design these products for increasingly longer lifetimes [132]. However, as products experience longer life cycles, they should be adaptable to changing requirements [66].

Because the risks and design investment are high, aerospace families tend to be product driven redesigns. One example of this kind of flexibility occurring in the aerospace industry comes from Holmes study of platform design in medium lift helicopters [67]. It details the upgradeability of the Sikorsky S-70/UH-60 and how it has been able to adapt to new customer requirements due to the flexibility in its baseline architecture.

In addition to aerospace families being flexible for long life cycles, there is a recent trend to make reconfigurable products. A reconfigurable system is designed to maintain a high level of

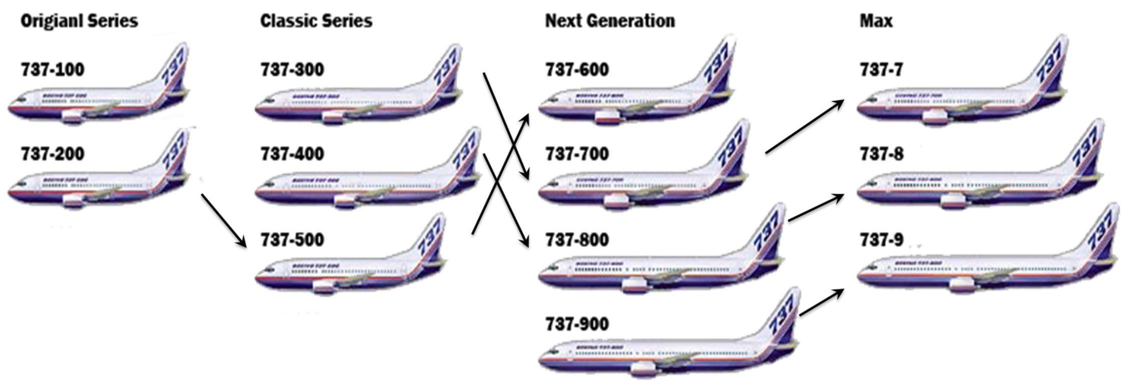


Figure 2: Boeing 737 Family

performance by changing its configuration to meet multiple functional requirements or a change in operating conditions[112]. This results in the system being able to perform multiple functions over time but not concurrently, to adapt into future configurations, and to still partially operate despite some component failures[46].

There is much ongoing research considering the use of Unmanned Air Vehicles (UAVs) as reconfigurable families [115]. This is because they do not have the same constraints as traditional manned aircraft on geometry, structures, and pilot survivability**.**

Observation 1

Aircraft have long life cycles and need to be flexible to changes in requirements. One method to remain flexible is by modularizing components to make the aircraft easily reconfigurable.

The virtues of long-life cycles and reconfigurability are well illustrated by the examples of the Boeing 737 and the F-35 Joint Strike Fighter (JSF).

1.1.3.1 Boeing 737

The Boeing 737, figure 2, is an example of a family driven redesign approach typical with the kinds of long life cycle designs characteristic in the aerospace industry. The 737 has had three different sets of products: the Original series, the Classic series, and the Next Generation series. Currently, there are plans to produce another derivative generation dubbed the MAX and is to be introduced in 2017.

Within each series there are a variety of product variants each targeting different passenger and range requirements. However, most of the variants share common wings and employ fuselage plugs which act as a scaling platform between the variants. A scaling platform uses scaling design variables to stretch or shrink the platform. For example the fuselage plug increases the length of the fuselage without affecting the diameter. Scaling platforms can be considered a subset of module-based platforms[52].

The current series the Boeing 737 Next Generation has four variants 600/-700/-800/-900. Recently Boeing announced that it would produce another series of 737 instead of a new "clean sheet design." This Boeing 737 MAX series will use a larger, more efficient engine and have a slightly modified 737 Next Generation airframe with the same fuselage lengths and door configurations. There are three variants planned to be part of the MAX series: 737 MAX 7, 737 MAX 8, and 737 MAX 9 to replace the 737-700, -800, and -900ER.

1.1.3.2 F-35 Joint Strike Fighter

The F-35 Joint Strike Fighter (JSF) is an example of a military aircraft family. The program was established in 1994 by the Secretary of Defense with the objective of bringing the different branches of the military (Navy, Air Force, and Marine Corps) together to collaborate on future strike warfare concepts[39]. Military planners hoped to reduce costs by maturing advanced technologies, components, and processes. The F-35 family, figure 3, is comprised of three variants: F-35A conventional take off and landing (CTOL), F-35B short take off and vertical landing (STOVL), F-35C carrier based (CV).

> Each branch, along with the United Kingdom royal Navy incorporated different requirements to the next generation fighter they were seeking to replace [152]:
>
> • Navy: A CV for first-day-of-the-war, survivable strike fighter to complement the F/A-18E/F
>
> • Air Force: A CTOL multirole aircraft (primary air-to-ground) to replace the F-16 and A-10 and to complement the F-22
>
> • Marine Corps: A STOVL aircraft to replace the AV- 8B and the USMC F/A-18
>
> • United Kingdom Royal Navy: A STOVL aircraft to replace the Sea Harrier.

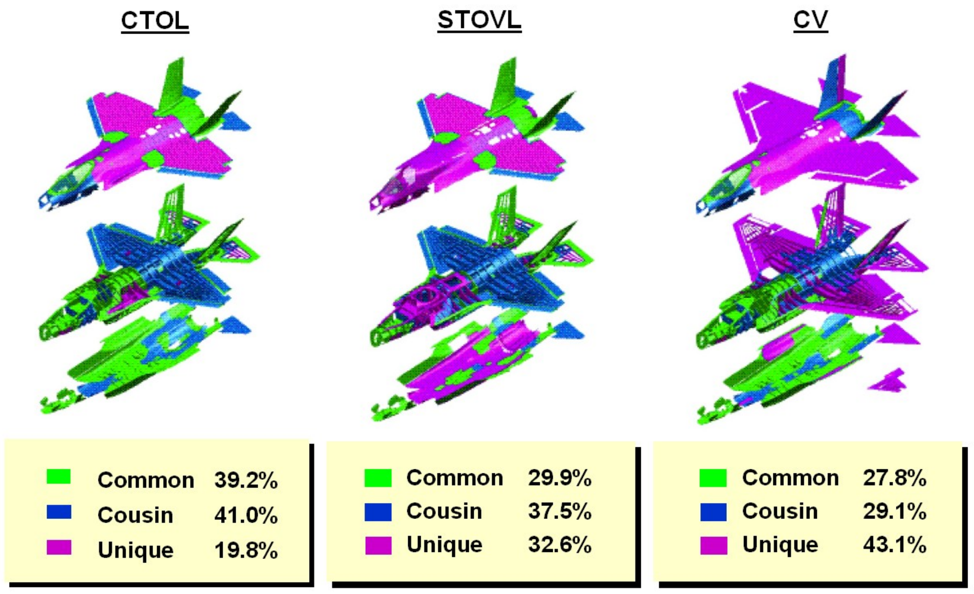


Figure 3: F-35 Commonality [56]

The canopy, radar and most of the avionics are common to the three variants [1]. The marine variant of the JSF is very similar to the air force variant but with a slightly shorter range because some of the space used for fuel is used for the lift fan of the STOVL propulsion system.

The main differences between the naval variant and the other versions of the JSF are associated with the carrier operations. The internal structure of the naval version is very strong to withstand the high loading of catapult-assisted launches and tailhook arrested landings. The aircraft has larger wing and tail control surfaces for low-speed approaches for carrier landing. Larger leading edge flaps and foldable wingtip sections provide a larger wing area, which provides an increased range and payload capacity.

While still in development, The F-35 total development funding is now estimated at $56.4 billion to be completed in 2018. This is a 26 percent cost increase and a 5-year schedule slip from the current baseline established in 2007. Affordability for the U.S. and its partners is challenged by a near doubling in average unit prices since the program started and by higher estimated life cycle costs. Going forward, the JSF requires unprecedented funding levels in a period of more austere defense budgets [156].

The STOVL variant has significant technical problems and deficient flight test performances. To address these technical problems and test deficiencies of the STOVL variant, the DOD significantly scaled back its procurement quantities and directed a 2-year period for evaluating and engineering technical solutions to inform future decisions on this variant. DOD also “decoupled” STOVL testing from the other two variants so as not to delay them and to allow all three to proceed at their own speeds. Lockheed Martin must turn around the F-35B within two years or expect its termination [19].

The difficulties experienced with the F-35 program are not uncommon with the development of new aerospace products. However, this program does reinforce one key observation about product families.

Observation 2

If products requirements are too divergent, the case for common components becomes weaker.

1.2 Product Family Design Challenges

There is a fundamental trade-off in product families, offering variety to satisfy the needs of different customers versus enabling cost savings through component commonality, figure 4. If there is no commonality between the different products, each design is independent and may result in higher family costs due to duplicated effort. If every component is common, then there is only one unified design that may have high costs due to added complexity and may compromise performance to such a degree as to no longer be able to satisfy customer requirements.

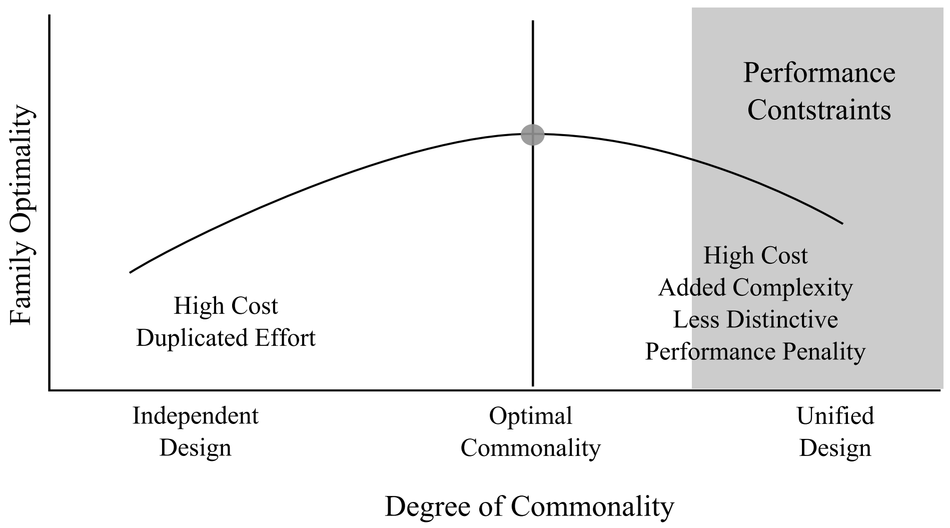


Figure 4: Fundamental Family Trade-off

Observation 3

Platform specification trade-off between product performance and cost should be carefully and fully investigated. Determining the optimal extent of component sharing between products is paramount for the product family to be successful.

In product family design there are three critical areas of study [89]:

1. Market analysis is required to identify the appropriate family architecture, to estimate the number of products to be manufactured, and to specify the requirements for each product.

2. Platform selection is necessary to determine the number of shared platforms and how these platforms are used in the final products.

3. Design optimization is required to identify settings of the design parameters, to maximize the portfolio profit, and to minimize the change in each product's performance from their individual ideal given all of the family constraints.

In the generalized product family design problem the platform configuration is not known. This results in many discrete options to define all the possible configurations and leads to a large combinatorial space.

The large combinatorial nature can limit the usefulness of traditional optimization algorithms because the space can be non-convex and filled with many local optima. Most combinatorial optimization problems are NP-hard. This means there is not an efficient algorithm that can be used to solve this type of problem within polynomially bounded computation times [3], ergo the time it takes to solve the problem does not scale polynomially with the number of variables. Also due to this combinatorial space, it is also too computationally expensive to evaluate every possible alternative.

In addition to these three critical areas of study, product family design can be divided into four approaches, figure 5 inspired from Jiao et al[72]. These approaches depend on if the product family is a new design or based around existing products and components and if the design is product or family driven. Product driven approaches tend to design products sequentially with subsequent designs using components designed for the prior products. Family driven approaches tend to design products in parallel. The risk associated with the different approaches increases from the product driven redesigns to family driven new designs. This is because in new family driven designs there is less known at the start as well as more than one product being developed, increasing the initial investment and resources required for the design. However, because production has not started, new family driven approaches have the potential to return a better family because platforms are optimized given all the products' constraints.

Another factor that can complicate product family design is when the products operate cooperatively, such as when several different aircraft perform a set of interrelated missions. In this example, the capabilities of different aircraft can be traded against each other while maintaining the same overall mission effectiveness. Some of these trades in a surveillance mission could be increasing aircraft detection range, higher speed, or increasing the number of aircraft.

Analyzing these kinds of system-of-systems (SoS) problems brings several additional challenges. Because the systems are connected, models need to be created that accurately reflect the behaviors between the systems. The rules governing the behaviors may be difficult to know and encode yet are pivotal to the analysis. For example, in a given mission there could be several different sequences of tasks to accomplish the mission. Finally when these systems interact, there can be an overall emergent behavior in how the systems perform. Additionally, a SoS problem is inherently a stochastic process where a given set of inputs yields a distribution of outputs, rather than a single

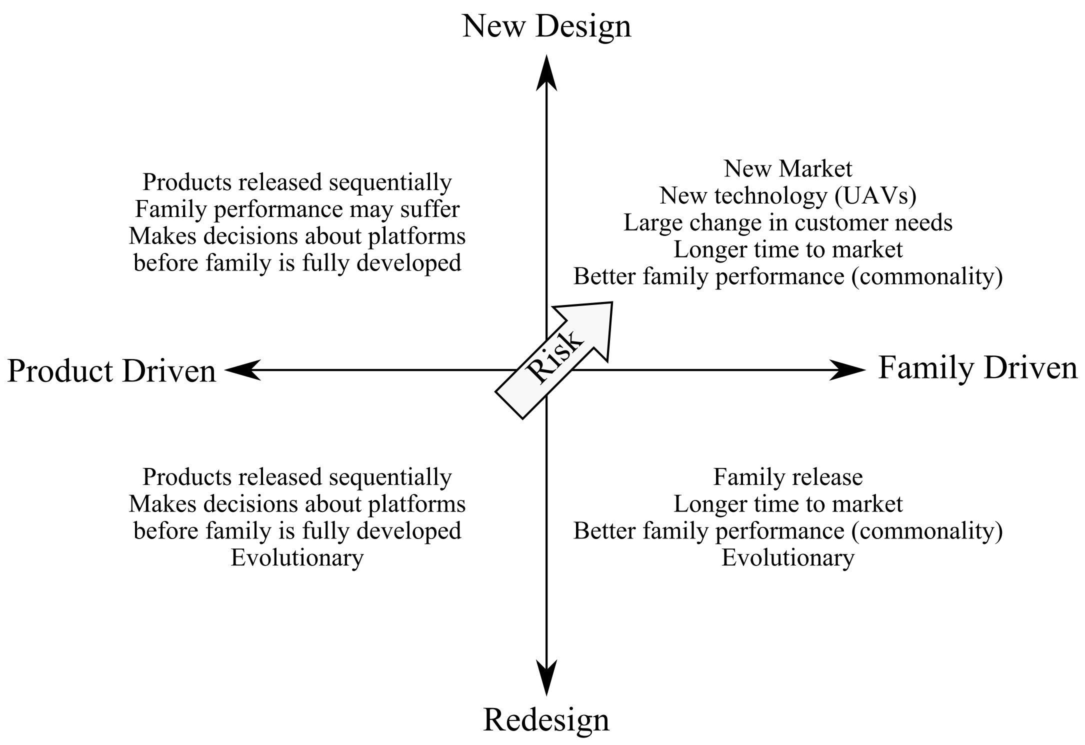

New Design
Products released sequentially
Family performance may suffer
Makes decisions about platforms
before family is fully developed
New Market
New technology (UAVs)
Large change in customer needs
Longer time to market
Better family performance (commonality)
Product Driven
Risk
Family Driven
Products released sequentially
Makes decisions about platforms
before family is fully developed
Evolutionary
Family release
Longer time to market
Better family performance (commonality)
Evolutionary
Redesign

Figure 5: Product Family Approach

value. The exploration of the SoS design space needs to be able to capture the mean trend as well as the output distributions.

The model's ability to capture these interactions raises the overall complexity, and results in increasing the analysis time. This increase in time is compounded by an increase in the number of cases necessary to capture the output distributions. These models are computational demanding and further complicate an already difficult product family design problem. Exploration of a SoS design space is an active area of research, and critical for a system-of-systems product family design.

1.3 Research Focus and Overview

This section concisely establishes the link between key research questions and hypotheses as well as provides an overview of the experiments performed. It is understood from the introduction to product families that component commonality is paramount to determining a good family. The main research question is derived from observation 3.

Research Question 1

In a product family design problem that has system-of-systems elements, how can commonality information be better incorporated to identify the relationship between product performances and cost?

To begin to answer this primary research question chapter 2 surveys current product family design methods. Based on the literature review, there are opportunities for a new product family method to solve product family design problems when commonality is not predetermined and all the design parameters are unknown. Problems with unknown commonality have the potential to yield higher performing families but are more difficult to solve because both the platform configuration and design parameter settings are unknown, figure 5. This leads to the main hypothesis that a new product family design methodology is needed.

Hypothesis 1

Incorporating knowledge about potential family platforms into a larger product family design methodology can focus computational resources away from considering poor platforms, making the design process more efficient, and helping to identify the ideal trade-off between product performances and portfolio costs.

Chapter 3 formally introduces a generic product family design methodology based off of the Generic Integrated Product and Process Development (IPPD) as a framework. Hypothesis 1 is difficult to test but can be decomposed into two sub-hypotheses.

Sub-Hypothesis 1.1

If poor component combinations can be eliminated from consideration, then design resources can focus on identifying the ideal trade-off between product performances and portfolio costs.

This sub-hypothesis is easily tested with a simple thought experiment. In the case of the general product family design problem the ideal sharing is not known and that the number of combinations to investigate can be prohibitively high. As with other design problems with high dimensionality, a screening test reveals which design parameters would most affect the solution. In an analogous way for a discrete space, pruning poor choices frees more computational resources for analyzing the remaining important options. The other sub-hypothesis 1.2, relies specifically on observation 3.

Sub-Hypothesis 1.2

If combining components that are more similar makes a better family platform, than combining dissimilar components, then a method to extract these patterns will aid in the formulation of the family.

This sub-hypothesis leads to an additional research question.

Research Question 2

How can family platform opportunities be systematically identified?

Existing literature comments that simultaneous optimization of the product family subject to the platform configuration and all of the design variables can lead to better performing family. Many of these methods also require making decisions about the platform configuration to lessen the combinatorial explosion of options. Unfortunately, if these decisions prune good options, then the resulting family will be less than ideal.

The product family hierarchy, figure 14, suggests that points that are in close proximity to each other in a given subspace would form a better platform than if they were grouped with points that were more dissimilar. Methods to extract these similarities must be found. To that end, two more hypotheses are put forward centering around alternative approaches. The first explores capturing similarity using a data mining approach, chapter 4.

Sub-Hypothesis 2.1

If a sufficiently accurate and dense database can be generated, then a machine learning pattern recognition technique like fuzzy clustering could be used to help identify component commonality and potentially form a platform.

The major challenge with the fuzzy clustering approach is that the design database is sufficiently dense in the feasible design regions. Because this requires a large number of cases to be generated, this limits its applicability to fast executing design codes. To address this scalability limitation, the other approach also attempts to capitalize on all of the evaluated points by creating a surrogate model of the design database's joint probability distribution.

Sub-Hypothesis 2.2

If a model can be generated that encodes the joint probability distribution, then component similarities can be inferred given performance constraints.

From the study of example aerospace families, products with long life cycles will need to meet changing requirements, observation 1. In other words, for a successful aircraft family the aircraft should be flexible to changing requirements. This then leads to an additional research question.

Research Question 3

How can these two approaches capture the family commonality sensitivity to changing requirements?

The clustering approach effectively extracts component commonality relationships from feasible subsets. Alternative requirements like those that may be placed on the family in the future then will yield a different subset for the clustering approach to process.

Sub-Hypothesis 3.1

If the design constraints are changed, then using the new feasible subset from the design database and performing the pattern recognition will reveal the sensitivity of component sharing.

The probabilistic approach extracts component commonality relationships through the comparison of posterior distributions of design variables. Using statistical inference, these posterior distributions are conditioned to the relevant requirements.

Sub-Hypothesis 3.2

If the design constraints are changed, then any changes to the posterior distributions from the probabilistic model will reveal the sensitivity of component sharing.

This generic product family design methodology can utilize either of the proposed approaches to help guide the family commonality exploration and selection. Although the generic methodology cannot be tested directly, several experiments are performed to test the various hypotheses.

Experiment 1 tests the effectiveness of the fuzzy clustering approach by comparing the equivalent hierarchy and family platform variable estimates to a baseline electric motor study. Using the platform configuration from the baseline study, the fuzzy clustering approach can give a statistical estimate of the platform study. The general accuracy of the fuzzy clustering approach is then assessed by comparing the estimated platform variables to those of the baseline. If the method is unable to recover the values then sub-hypothesis 2.1 can be rejected, provided the method was initialized with a sufficiently accurate database.

Experiment 2 tests the robustness of the fuzzy clustering approach and the required accuracy of the design database for the motor family by generating the database using three different methods: a Latin Hypercube design of experiments (DoE), a uniform Monte Carlo exploration, and an evolutionary algorithm. Additionally, the number of design cases is varied for databases generated using the Latin Hypercube DoE and the Monte Carlo exploration. Increasing the density of points in the database should enable better estimates of the platform design variables as there are more points closer to the baseline study.

Experiment 3 uses a different electric motor design study to test sub-hypothesis 3.1. In general, flexibility can be seen as meeting alternative sets of requirements. This then can be translated into constraints that yield a new feasible subset. So by using a different study with different requirements the design cases that violate these new constraints are removed. This new subset is then processed using the clustering and pattern recognition technique to determine the platform configuration changes.

Experiment 4 builds a Bayesian network model of the design database using data generated from a multiobjective genetic algorithm for the electric motor. This experiment supports sub-hypothesis

2.2 by again using the baseline electric motor study from Experiment 1. Furthermore, the output of the probabilistic approach can be benchmarked against the fuzzy clustering approach.

Finally an aircraft family design problem with system-of-systems elements is considered to demonstrate the entire proposed generic product family design methodology. This demonstration problem involves two scenarios each with two interacting types of aircraft. One scenario is an aerial firefighting using two types of unmanned aircraft: one patrolling for fires and one to extinguish. While the other scenario is a maritime surveillance set of unmanned aircraft performing missions that include search and rescue, polluter identification, and oil spill response. Both the fuzzy clustering approach and probabilistic approach are used to attempt to identify similar components.

1.3.1 Outline

Chapter 1 : Introduction Introduces the importance of product families and supplies a few examples relevant to the aerospace industry. These examples allow a few key observations to be made that help drive this research.

Chapter 2 : Design Method Review Presents a review of the current design methods that are used for product families problems. This chapter also identifies the challenges encountered in designing product families and notes a few opportunities of a new method. This chapter reiterates the main research questions and main hypothesis.

Chapter 3 : Product Family Methodology Proposes the integrated product family design methodology using the Generic IPPD design process as the framework. In the proposed methodology the component commonality can be captured with either approach.

Chapter 4 : Clustering Approach Formulation Introduces the theory behind using the pattern recognition technique of fuzzy clustering and arrives at hypothesis 2.1 and hypothesis 3.1. The importance of the design database accuracy and the scaling limitation of this approach are also noted.

Chapter 5 : Probabilistic Approach Formulation Introduces probabilistic graphical models and how they may be employed as a probabilistic surrogate model hypothesis 2.2 and by extension hypothesis 3.2. The background supporting the use of Bayesian networks is put forward along with a few other applications of Bayesian networks.

Chapter 6 : Probabilistic Approach Feasibility Presents a study of the feasibility of using the Bayesian network as a surrogate model. First a small optimization was run to test the network's reliance on the kernel basis functions. Another example is presented showing the ability of the Bayesian network to capture the design space of the Breguet range equation.

Chapter 7 : Electric Motor Family Study Presents the common family design problem of universal electric motors. Experiments 1-3 tests various aspects of the clustering approach. Experiment 4 tests the efficacy of the probabilistic approach.

Chapter 8 : Aircraft Family Study Demonstrates the product family design methodology on the design of a family of aircraft. The models and challenges necessary to analyze the family are discussed. The commonality approaches are tested by comparing an expected commonality change between two different feasibility criteria.

Chapter 9 : Summary and Contributions Summarizes the work performed for this dissertation and recounts the contributions made.

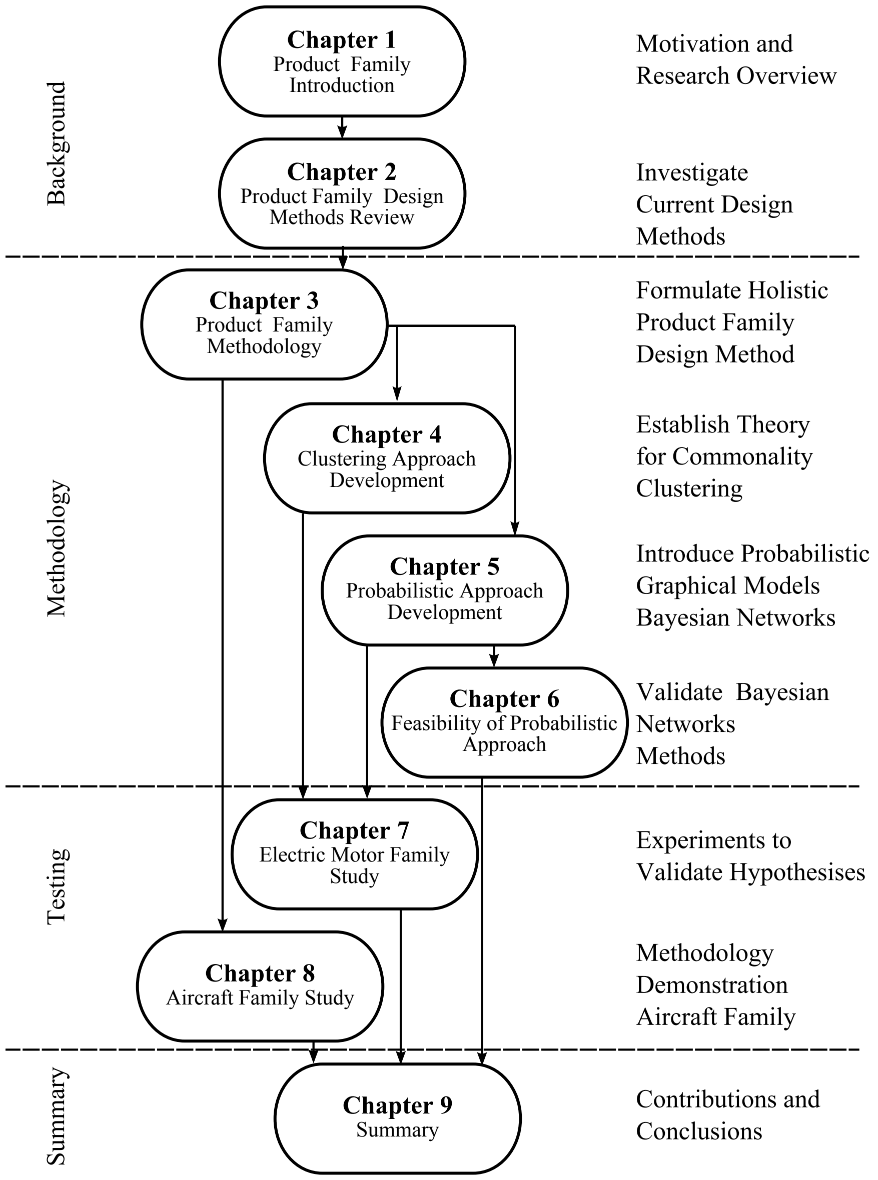

Background
Methodology
Testing
Summary
Chapter 1
Product Family Introduction
Motivation and Research Overview
Chapter 2
Product Family Design Methods Review
Investigate Current Design Methods
Chapter 3
Product Family Methodology
Formulate Holistic Product Family Design Method
Chapter 4
Clustering Approach Development
Establish Theory for Commonality Clustering
Chapter 5
Probabilistic Approach Development
Introduce Probabilistic Graphical Models Bayesian Networks
Chapter 6
Feasibility of Probabilistic Approach
Validate Bayesian Networks Methods
Chapter 7
Electric Motor Family Study
Experiments to Validate Hypothesises
Chapter 8
Aircraft Family Study
Methodology Demonstration Aircraft Family
Chapter 9
Summary
Contributions and Conclusions

Figure 6: Thesis Overview

Chapter II

CURRENT PRODUCT FAMILY DESIGN METHODS

The previous chapter establishes the benefits of product families and the trade-off that is made by sharing components. Observation 3 comments that the fundamental trade-off in product families stems from the commonality decisions of the family. This observation drives the following primary research question.

Research Question 1

In a product family design problem that has system-of-systems elements, how can commonality information be better incorporated to identify the relationship between product performances and cost?

To begin to answer this question, the state of the art methods in product family design need to be understood. This chapter surveys the different current methods utilized at the various stages in product family design. First a description is given for product design, then a mathematical overview of the combinatorial issue of platform configuration in product family design. Next various methods for approaching the different classes of product family design problems are considered. Then a few visualization approaches for families are considered that help designers understand the trade-offs involved in platform specification. Finally a few gaps are noted that establish the need for an additional design methodology.

In designing a single product there are several processes that must occur . Figure 7 shows an overview of the general processes of product design. At the start, a market analysis should be performed to better establish target specifications that the product must meet to satisfy customer needs. Without a thorough assessment of the consumer market, correct design requirements cannot be accurately specified. If the set of requirements passed to designers fails to satisfy customers, then the product is doomed to failure. Once requirements are identified, designers can begin to formulate different potential product architectures.

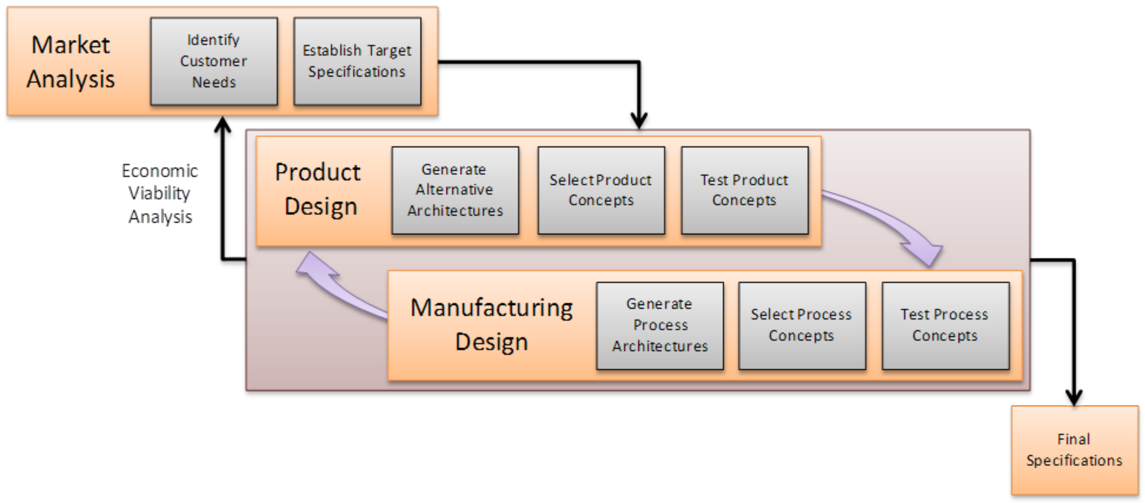


Figure 7: Product Design

Each architecture can be radically different from one another and must be traded against each other until a tractable number of alternatives remain for additional analysis. The alternatives can then be further explored with higher fidelity, eventually including the interactions between the product design and the manufacturing processes. As more information is collected about each alternative, it is possible to down-select to the best economically performing candidates. This process is iterated until the product is both technically feasible and economically viable. Ultimately, success depends not only on the final product but also on designing for the right market.

Like the process for product design, family design combines business and marketing strategies with engineering ranging from technological aspects of design to manufacturing and support. The difference now is the coupling of the products together through the use of platforms[121]. Figure 8 provides a holistic view of the fundamental product family design process inspired from Jiao [72]. Similar to axiomatic design from Suh [155], the process can be viewed as different coupled design domains with mappings across them.

Jiao, Simpson, and Siddique describe these separate domains [72]:

> Customers Needs (CNs) Domain characterizes groups of customer needs representing markets niches products should target. These niche requirements then cascade trough the other product family domains
>
> Functional Requirements (FRs) Domain translates the CNs into the set of engineering

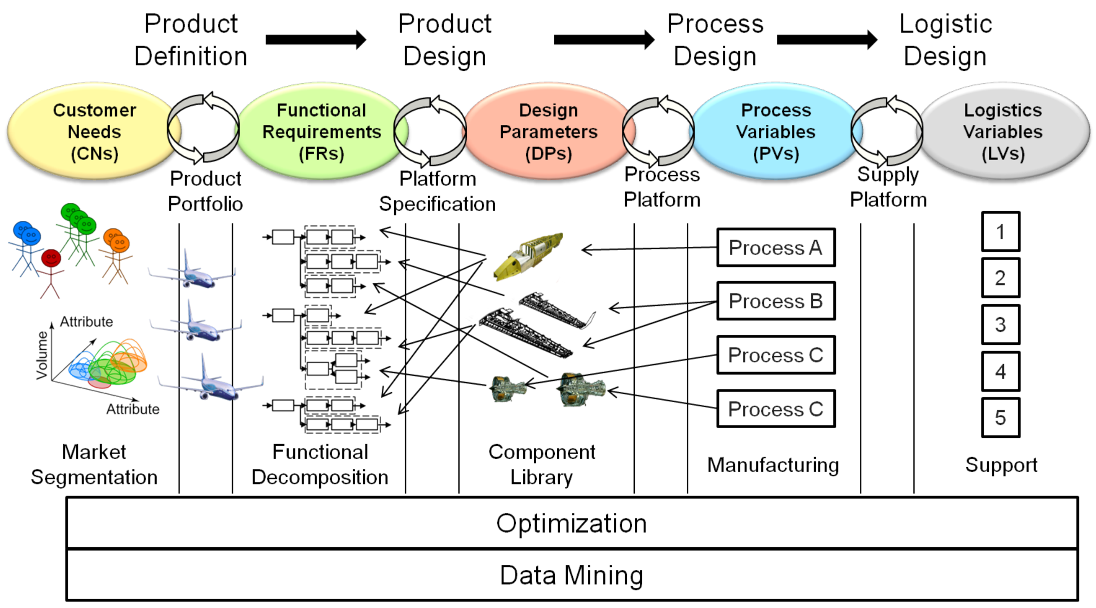


Figure 8: A Holistic View of Product Family Design and Development

requirements that the different product designs must meet

Design Parameters (DPs) Domain determines the physical design parameters and product platforms to meet the FRs

Process Variables (PVs) Domain contains the manufacturing and production planning, tooling, setup, equipment details associated for the product assembly

Logistics Variables (LVs) Domain addresses issues related to supply chain with details into resource allocation and supplier management

The mapping between the domains is as important as the domains themselves. For example, the product portfolio is the mapping between different products in the functional domain to each different market niche in the customer domain. Market segmentation occurs in the Customer Needs domain by deciding which customer needs are most similar and how many products need to be produced to capture the specific markets. The product platform specification is the mapping between the functional domain and the design parameter domain. This family platform configuration mapping is the primary focus for this research.

Family platforms are not only restricted to product components and can include manufacturing processes and supply chains to maximize the advantages of commonalities. It is important to note that all the different domains have feedback loops, manufacturing and logistics concerns could strongly affect the optimal commonality decisions. For example, the family may fail if a component is unable to be manufactured or if the supply chain limits access to critical materials.

The main stages in family design are [121, 72]:

1. Product Definition - Market segmentation and determining the product portfolio
2. Product Design - Platform specification and determining the values for all of the shared and unique component design variables
3. Process Design - Process planning and manufacturing concerns
4. Logistics Design - Supply chain design and maintenance issues

The supply chain is the process that connects all of the companies together from the raw materials to the delivery and maintenance of the final product. When the supply chain becomes a significant part of controlling operational costs it should be included as soon as possible in the design process[91]. However, this dissertation focuses on the earlier product definition and product design phases.

This is because coupling those domains with the upfront issues of determining product architecture, portfolio, and component platforms causes the design problem to become too complex. There are some methods that discuss process and logistic design but they are not the focus of this thesis work.

The studies in this literature review begins with a discussion on the combinatorial problem of family platform specification. Then current design methods are presented organized by the stage of family design on which they focus and by the general techniques they employ. Because of the primary focus of this research, most methods presented will focus on product definition and design with a few example studies on process and logistic design.

2.1 Combinatorial Configuration Complexity

Prior to analyzing alternative product family design methods it is important to understand the magnitude of the combinatorial problem at the heart of product family design. Specifically the configuration of the family platforms which specifies how components are shared between the products.

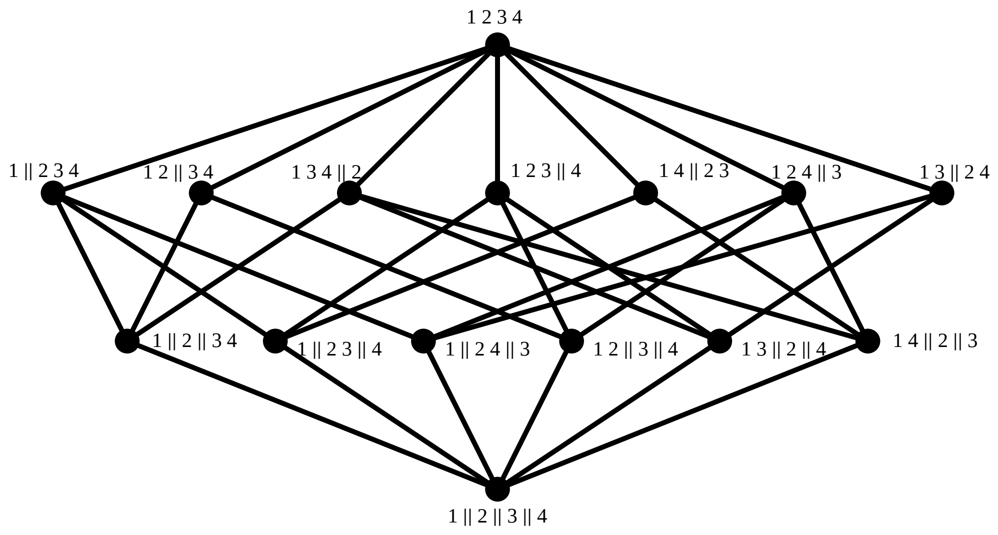


Figure 9: Hasse Diagram Showing $\mathfrak{p}$ the Partially Ordered Set of Partitions of $\{1,2,3,4\}$ [143]

The platform configuration platform can be formalized using set theory, where it is described as a "partitioning problem." A partition of a set is defined as a division of the set into non-overlapping and non-empty subsets. These subsets are collectively exhaustive and mutually exclusive with respect to the set being partitioned.

Sets can be partitioned many different ways. A Hasse diagram, figure 9, illustrates all the different ways to partition a set of four members. The nodes in the graph represent different unique divisions of the set. Edges in the graph show that one node is a finer partition of the connected nodes. A partition, p_1, is finer than another partition, p_2, if all of the elements in p_1 are subsets of p_2. For example, $\{1|2,3|4\}$ is a finer partition of $\{1|2,3,4\}$ but not $\{1,2,4|3\}$. The top graph node shows the coarsest partition where all four elements are together, and the bottom shows the finest partitioning where each element is by itself.

Using product family terminology this figure shows how a particular component could be shared across four different products. If products are in the same subset then they have a common component. To fully understand impact of the platform configures all 15 different options must be considered and evaluated. For each option, the optimum platform design parameters and product specific design parameter settings would have to be found. The number of combinations to evaluate grows rapidly as the number of products in the family increases. With some arbitrary number of products n the number of combinations is counted by the Bell number, Eq 1.

$$B(n) = \sum_{k=0}^{n} \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \begin{pmatrix} k \\ j \end{pmatrix} j^n \tag{1}$$

Product families are comprised of multiple modules, each with their own sharing space. The total number of combinations, T, of each module's sharing, multiply according to Eq 2

$$T = \prod_{m}^{|\mathfrak{M}_{\mathfrak{l}}|} B(n_m) \tag{2}$$

where $|\mathfrak{M}_{\mathfrak{l}}|$ represents the number of elements in the module space, and n_m is the number of products that could share the m^{th} module. For a large number of products with many modules, the platform configuration space becomes too large to enumerate all of the feasible combinations and fully analyze them. Most of these combinations are infeasible or dominated by better combinations of common components. Engineering intuition and company heritage are most commonly used to help down-select from all the different component sharing possibilities to only a few alternatives around which to perform a trade study. Now that the magnitude of the combinatorial platform configuration space is understood the next section starts to analyze current product family design methods.

2.2 Product Definition

The first step in product definition is market analysis. This entails describing the customer needs and identifying target market segments. These processes are coupled with the product positioning problem which focuses on increasing variation and diversity in the offered products. Product positioning attempts to assign the strategy of products to market segments and can possibly include understanding competing firms and the evolution of the products overtime.

Data mining methods like fuzzy clustering, heuristic search algorithms, and conjoint analysis have been increasingly used to solve the family positioning problem [121]. Data mining methods are methods that typically employ historical data sets to identify and group common elements. For example, a historical database of customer requirements can be mined to partition the space and identify unique groups of customers known as market segments. One market segmentation method took a transactional database of functional requirements and customer needs, and it generated association rules using fuzzy relations generated from fuzzy clustering [170]. Another example of a market segmenting study was done by Kazemazadeh et al [78]. They use conjoint analysis to study how customers develop their product preferences. Then using quality function deployment (QFD), a cluster analysis identified groups for different segments. Their test case results in the sponsor company developing a product family rather than a generic product enabling a cost reduction along with increased customer satisfaction. Agard also performs a product positioning study using existing databases of industrial customer preference, part descriptions, and manufacturing processes to extract association rules for segmenting the market as well as standardizing family platforms [4]. Companies can significantly improve their future products by collecting and maintaining detailed information for use in developing the next generation.

After segmenting the market, customer needs are translated into functional requirements. Here the architecture is developed. A family architecture is concerned with modularity and functional breakdown and describes both the product architecture as well as the commonality between the different products [72]. Similar to family architecture, product architecture can be defined as the scheme that translates the functional elements of a product into physical components and the way in which these components interact [161]. Product architectures greatly influence optimal platform configurations and product variety. The idea of modules and modularity are central in constructing versatile product architectures [161]. Modules are independent parts of a product that can be treated as logical self-contained units with standardized interfaces and allow composition of products by combination [110, 103]. There are a variety of ways to implement functional requirements and an efficient functional breakdown allows for better standardization between the modules.

In considering a family it is important to analyze alternative architectures. There are instances of family problems where the exact requirements for individual products may not be known which further complicates the selection of family architecture and market segmentation.

For example, there could be multiple ways of accomplishing a particular mission like aerial reconnaissance. The customer could purchase a single aircraft with long range sensors or more aircraft with lower performing sensors. Both options may be able to accomplish that mission but there could be family implications as the number of aircraft are varied. Figure 10 shows the continuum

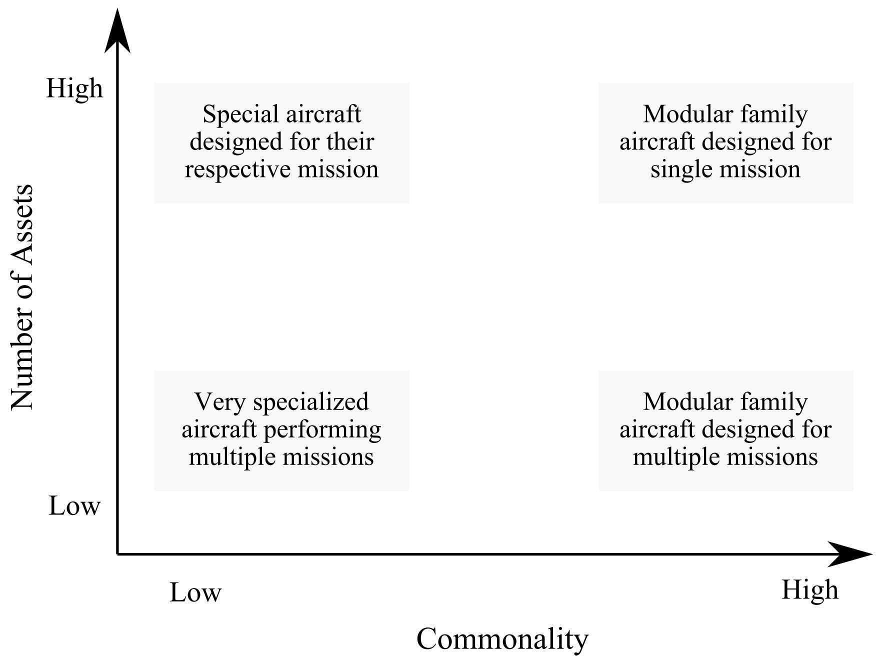

High
Special aircraft designed for their respective mission
Modular family aircraft designed for single mission
Number of Assets
Very specialized aircraft performing multiple missions
Modular family aircraft designed for multiple missions
Low
Low
High
Commonality

Figure 10: Continuum of Solutions

of family architectures potentially developed that trade-off product commonality and the numbers of products to be produced. This coupling of family architecture and product commonality further increases the combinations to consider when performing the family design. Analyzing this trade requires the development of system-of-systems models.

For a given architecture, a functional decomposition can take place when translating customer needs to functional requirements. There are several methods for translating functional requirements to modules which can be described by design parameters.

One method is articulated by Rai and Allada[122]. They use four steps in their module identification using functional decomposition. First, a functional representation generates the modules via the function architecting process and provides a rich description of the function-module mapping, modules and their relationships. Second, a Pareto-optimization generates the Pareto design solutions while satisfying all the design requirements through proper module configuration. Third, a higher-level decision making can be used to solve various problems related to modular designs using the information gathered from the Pareto optimal solutions.

Another method is proposed by Stone et al. [153]. Here their method introduces three heuristics to create a systematic module identification approach. A function flow may pass through a product unchanged, it may branch, forming independent function changes, or it may be converted to a different form. According to their study, the choice of which module to implement is not obvious and these rules help to keep modules easily identifiable with a particular function.

Moon, Kumara, and Simpson identify modules by using a fuzzy c-means clustering on the functional breakdown which represent all of the functions the product must do to meet the customer needs [104]. Effectively, they use data mining to determine product architectures by grouping components based on function chains.

Dabbeeru, Deb, and Mukerjee perform product selection by analyzing Pareto optimal designs and then perform a clustering to identify groups in the design space[32]. They go a step further and identify the lower dimensional manifold in which these potential products lie to help visualize the clusters in the design space. The design team can then use that information to decide which products constitute the product family.

Sichani divides family design problem into simpler subproblems to optimize[142]. Additionally, through the use of game theory, marketplace competition between two firms can be included.

Haubelt introduces hierarchical graph modeling to describe the product architecture[60]. This hierarchical model allows for efficient exploration of the flexibility and cost tradeoff curve. System flexibility is accounted for by ensuring the system is able to operate with a complete set of different applications. Similar to Haubelt, Umeda et al. develop a methodology to support generational families [163]. They do this by also developing a functional behavior state modeling scheme to identify an upgradeable design that can adapt to future requirements.

Once the product architecture and product modules are defined, the products can be codified by sets of physical design parameters. Modules themselves are described as subsets of the overall product design parameters.

2.3 Product Design

Product design occurs after the set of products and the product portfolio is described. The goal at this stage is to identify the component configuration and design parameters as well as all of the product specific design parameters. Common methods employed in this step use some sort of optimization algorithm to determine the design parameter settings to maximize the family performance while minimizing the cost.

Fujita suggests there are three classes of product design optimization problems: Class I, Class II, and Class III[51]. In Class I design problems, the design parameters are optimized given a fixed platform configuration. For example, knowing which components are shared and only optimizing each product based on the unique and shared components would be considered a Class I problem. Nelson et al. proposed for a multicriteria optimization formulation for Class I[109]:

maximize	*product 'performance'*
with respect to	*product design variables*
subject to	*product requirements*
	component commonality constraints

In a Class II family design problem, the design variables are fixed and the sharing decision variables select the product components from an existing library. Products are still optimized but under the discrete decision variables that constrain components to the predefined candidates:

maximize	*product 'performance'*
	component commonality
with respect to	*sharing decision variables*
subject to	*product requirements*
	component commonality constraints

Finally in a Class III design problem, both design parameters and the platform configuration are simultaneously optimized. Class III problems are the most difficult problem class because both sharing or design parameters are unknown:

maximize	*product 'performance'*
	component commonality
with respect to	*product design variables*
	sharing decision variables
subject to	*product requirements*
	component commonality constraints

Fellini formalizes the optimization mathematically for the product family design problem as maximizing the performances and commonality for each product p in the family $\mathcal{P}$[43]:

$$
\boxed{
\begin{aligned}
&\text{maximize} \begin{cases} f^p\left(\mathbf{x}^p\right) \\ \sum_{ijpq} R_{ij}^{pq} \end{cases} && \forall p \in \mathcal{P};\ p < q \\
&\text{with respect to} \quad \begin{matrix} \bar{\mathbf{x}} = \left\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m\right\} \\ R \end{matrix} && m = |\mathcal{P}| \\
&\text{subject to } \mathbf{g}^p\left(\mathbf{x}^p\right) \leq 0 \\
&\mathbf{h}^p\left(\mathbf{x}^p\right) = 0 \\
&R_{ij}^{pq}\left(x_i^p - x_j^q\right) = 0 \\
&R_{ij}^{pq} \in \{0,1\} \\
&(i,j) \in \mathbb{S}^{pq}
\end{aligned}
}
$$

where:

$\bar{\mathbf{x}}$ is set of all the product design variables;

$\mathbb{S}^{pq}$ is the design variables between product p and product q that are possible to share;

R_{ij}^{pq} binary relation variable indicating sharing design variables between products;

m number of members in family $\mathcal{P}$.

Khajavirad [82] expands these problem classifications to account for restricted versus generalized commonality, figure 11. Restricted commonality limits platforms and allows them to be shared across the entire set of products, R_{ij}. There is a need for product family methods able to deal with the high computational costs of Class III problems with generalized commonality, R_{ij}^{pq}[82].

Of the three classes, Class III problems are the most difficult because the design parameter optimization process is coupled with the combinatorial problem of platform configuration. The problem complexity is further increased while considering generalized platform configurations in families with more than one possible platform. The advantage of Class III problems is that by simultaneously optimizing both design parameters and platform configuration, designers can find a potentially better family.

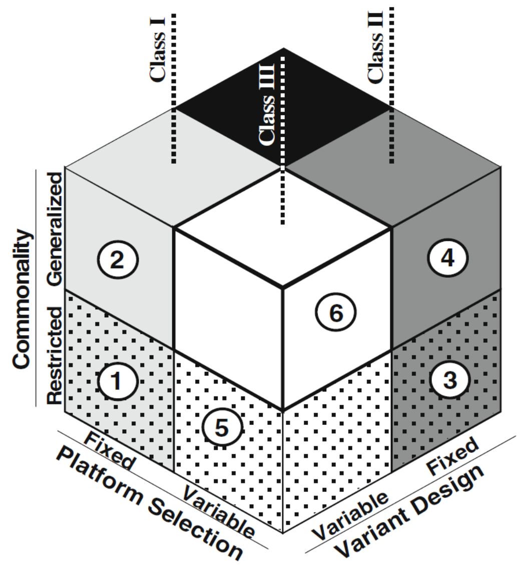


Figure 11: Classification of Product Family Optimization Formulations [82]

2.3.1 Commonality

Frequently, in forming the optimization problem, the objective function incorporates commonality indices. Commonality indices are used as a surrogate for direct calculation of cost savings based on commonality [83]. Although direct calculation of costs due to commonality are desired, some cost models are filled with rough estimations that only add to the difficulty of using them without gain in accuracy. Pirmoradi compiled six widely used commonality indices[121], table 1. The basic idea behind using these indices is to aggregate of the total the degree of commonality that is included for a particular platform configuration.

Unfortunately, at times commonality indices can be "misleading for large family assessments" or limited to particular platform types (scalable or modular) [121]. Again this is because the desire of a family is to reduce costs and these indices do not incorporate any cost estimates. The benefit of using commonality indices instead of cost estimates is there can be significant error in cost estimates especially during the conceptual design phase when the family would be architected. Selection of a commonality index unsuited for the family being optimized can negatively impact overall performance.

Table 1: Characteristics of Six Widely Used Commonality Indices [121]

Index	Conditions of use	Limitations	Improvement area(s)
DCI, $CI^{(C)}$	• Component-based ($CI^{(C)}$ considers process commonality too) • Moving boundaries • $CI^{(C)}$ is an extension of DCI, taking factors such as product volume, and component costs into consideration	• Difficult to interpret due to varying upper limit for different families • Difficult to assess improvement of redesigns • Not useful for large number of options • $CI^{(C)}$ requires cost estimation which might be challenging	• Taking other attributes such as functionality , type of materials, and manufacturing processes into account • Allowing variety • Improving cost estimation approach
TCCI , CI	• TCCI is a normalized version of the DCI • Component-based • Fixed boundaries (0-1)	• Focusing only on the percentage of common/unique components rather than cost factors • Not capable handling large number of options	• Weighing components • Considering variety in addition to commonality • Using more clarification in criteria for differentiation
PCI, %C	• Component-based • Fixed boundaries (0-100) • %C is applicable to platform commonality measurement	• Unable to capture effect of component costs on commonality • PCI assumes known degree of differentiation	• Broadening the areas of consideration such as differentiation degree • Clarifying differences among differentiating and non-differentiating components
CDI	• Fixed boundaries (0-1) • Assessing both commonality and diversity • Allowing analysis in function, component, and family levels	Ignoring the information about component costs, materials, and processes	• Integrating higher architecture levels • Using further criteria such as cost, and manufacturing issues
CMC	• Fixed boundaries (0-1) • Considering the effect of each component on the overall commonality level • Allowing desired variety/commonality	More information-sensitive than other indices	Taking component performance into account
TCM	• Using GBOM for incorporating all possible variants into consideration • Using past data about selection probabilities of variants	• Applicable only if a GBOM is available • Difficulty in data collection • Better for assemble-to-order systems	Integrating process commonalization studies into the index

2.3.2 Optimization Approaches

This section describes several direct optimization approaches for solving product family design problems. Identifying shared common components in a product family is not a trivial task and is the subject of many studies. Two approaches that start using set theory come from Siddieque and Newcomb. Siddique and Rosen develop a Product Family Reasoning System (PFRS) to treat the combinatorial nature of configuration design at all levels in a product family which is comprised strongly from set theory [143]. Newcomb et al[110] also rely on the use of set theory to describe modularity in product families. The use of set theory as a starting point helps to create a robust framework for computation.

The Product Family Extension Configuration Design (PFECD) put forward by Zhao et al. is another approach to improve the efficiency of configuration design. PFECD is in the initial stage of development but attempts to solve contradictory problems in configuration structure using a knowledge base and rule system [171].

Once the platform configuration is decided in Class I or in the case of Class III problems remaining coupled, optimization is used. Product family design methodology studies typically make some sort of platform configuration assumption a priori to decrease the complexity of the problem[72]. These assumptions typically restricting them to Class I problems or a set of Class I problems if a trade study is being performed on two or more competing sharing configurations. Methods focusing on Class III problems will restrict the commonality so platforms can either be used by every product in the family, or not at all. This restriction is done to limit the combinatorial space so the problem is tractable.

Variation-Based Platform Design Methodology (VBPDM) is an example of a two stage method that picks the platform configuration by comparing variation of the input variables then uses variables with low variation as the platform across the whole family[107]. The second stage uses that platform and optimizes the family.

However, there have been several papers that try to analyze a product family while allowing the platform configuration to change. Methods that simultaneously optimize component sharing and design variable settings have the potential to find better families because product subsets may be more similar to each other than to other subsets of products. The majority of methods that attempt to solve Class III problems use stochastic methods, either genetic algorithms, simulated annealing, or particle swarm. Most of the product family design methods that try and solve the Class III problem use multiple stages[144]. In these kinds of methods, the first stage identifies the platform variables then simplifies the problem to a Class I problem. Messac et al. mentions a single stage approach is better because two stage approaches tend toward suboptimal solutions[98].

Some methods also approach Class III problems from a Multidisciplinary Optimization (MDO) perspective. In multidisciplinary optimization the system to be optimized is decomposed into subsystems which are typically different disciplines. Coupling variables link the different subsystems together transferring information.

Ferguson uses a multilevel MDO approach to optimize a reconfigurable racecar for different conditions [45]. This MDO approach is successful in finding the reconfigurable racecar performance improvements over the baseline racecar. For the top level optimizer, a genetic algorithm is used to determine the values for the coupling platform variables. However, this approach though was locked and considered a set of Class I problems because each MDO was done with the platform configuration fixed between the different scenarios.

Another approach for solving Class I type problems is Analytical Target Cascading (ATC). In ATC top level design targets are cascaded down through the hierarchy of models[102]. This is done with four steps i) specify the overall product targets, ii) propagate the product targets to the system and component levels, iii) design the system and components to achieve their respective subtargets, and iv) verify the top level targets are met. Kokkolaras et al. extend the ATC methodology for Class I problems[86]. Here separate sub-problems come from each product and commonality is enforced by using subsystems with common parents.

Khajavirad and Michalek[81] demonstrate a multistage gradient based method for solving the Class III joint platform selection and design parameter optimization problem. This method restricts component commonality to either be entirely shared or entirely unique across the family.

Khire et al. [84] introduce Selection-Integration Optimization (SIO) which integrates the platform configuration selection and design parameter optimization to be able to solve restricted Class III problems. SIO uses a Variable Segregating Mapping Function (VSMF) to transform the discrete platform configuration problem into a continuous problem. However SIO requires platforms to be shared completely, or not at all.

Williams [168] extends the original Product Platform Constructal Theory Method (PPCTM) proposed by Hernandez [64] to be able to address uneven demand across the different market segments and include multiple objectives. However, determining which platform configurations are most advantageous is not guided by a systematic process and is left to the designer.

More recently, Chowdhury et al. [28], building on ideas from SIO, created the Comprehensive Product Platform Planning (CP^3) framework. CP^3 formulates the generalized commonality product family design problem as a Mixed Integer Non-Linear Programming (MINLP) problem. The framework combines another mapping function to convert the discrete platform configuration problem into a continuous domain, while allowing platforms to be formed across any subset of products. A cost function was also implemented to account for platform sharing of different component complexity. A particle swarm optimizer was used because stochastic algorithms are better behaved on the multimodal product family problem than gradient based algorithms.

Genetic algorithms have also been used to address the joint family optimization problem where platform sharing is not initially specified by modifying the chromosome string to include sharing information. Multi-Objective Evolutionary Algorithms (MOEAs) have been demonstrated to address the high dimensionality of the single stage computational challenges [139]. Simpson [149] uses a genetic algorithm approach from the restricted commonality Class III problem, requiring platforms to have 100 percent commonality for the whole family. Simpson notes that seeding their initial population yields a richer Pareto set in fewer generations. Ölvander, Tarkian, and Feng also employ the use of the NSGA-II with a traditional chromosome string of both discrete (to control sharing) and continuous design variables, to find a Pareto optimal front showing the tradeoff between the degree of commonality in a family of industrial robots [114]. They note that including the degree of commonality in the objective function creates difficulty for the algorithm. Difficulties that are compounded when discrete combinatorial commonality variables couple with the continuous design variables.

To be able to solve generalized Class III problems, Khajavirad [82] uses a generalized 2D commonality chromosome with modified crossover and mutation operators to allow subsets of products to share components. Their changes were implemented on top of NSGA-II from Deb and can find a Pareto optimal set of solutions in a reasonable time [37]. The top level GA finds the optimal platform configuration while each lower GA optimizes a subset of products. When considering more products, the 2D chromosome representation scales better than previous approaches that only use a single level.

Shah et al. uses the ε-NSGA-II algorithm to explore the design space of a family of general aviation aircraft and then use visual analytics to facilitate the interaction between the designers and the optimization algorithm [139].

Dabbeeru, Deb and Mukerjee have a method employing the NSGA-II to find the Pareto frontier and then find its embedding in the lower dimension manifold[32]. They then use the embedding and specify a number of clusters to partition the objective space. By then inspecting the cluster centers they can find a set of target product performances for the family that would equally cover the performance space.

One potential problem with genetic algorithms for large Class III product family problems with generalized commonality is the chromosome. In the genetic algorithm metaphor, the chromosome is a collection of all the design parameters. With generalized commonality in product families, sharing variables are added to the chromosome. Effectively these act as switches that when evaluated tell the product models which parts of the chromosome is active for that case. Back in the genetic metaphor, these duplicate genes are called allele and only the dominate trait surfaces. Over many generations due to “genetic drift” the deactivated or recessive allele are passed to future generations but without contributing to the goodness. If suddenly the sharing activates the allele the performance of that point could be quite poor while the remainder of the products are evaluated on the basis of quality genetic material. Due to the breeding selection in genetic algorithms, the point will probably be dropped from the gene pool because of the poor performance from a recessive trait suddenly being active after many generations of drift. Loss in population fitness due to genetic drift has a direct effect on the performance of the genetic algorithm[130]. To alleviate any problems from genetic drift larger populations should be used or a modification to the selection process may be needed[130].

2.3.3 Visualization / Hybrid Approaches

In addition to pure optimization methods, there are studies that employ data mining for the platform specification problem which then may be coupled with some optimization to solve for the design parameter settings. With pure optimization, the final result can often be less informative than a design space exploration[92]. Including humans in the optimization process can improve the results of hard optimization problems[136]. While optimization is a useful tool, there are instances when the optimizer will fail to provide helpful information or arrive at an unusual point by exploiting some unknown constraints.

While humans can help guide the optimization, they also present difficulties. For humans to be helpful, they must be able to make an informed decision about the current state of the exploration. This is especially challenging for high dimensional problems like in product family design. To help humans understand these behaviors, several visualization techniques can be useful for presenting the bulk design space exploration data.

In family design visualization, the goal for the visualization is to showcase the similarity between the different modules for the different products. There are several applicable multivariate visualization techniques. Parallel coordinates is a visualization technique for embedding each point the high dimensional data set as line segment across several equally spaces axes [71]. Shah uses visual analytics, powered scatter plot analysis and parallel coordinates to help understand the results of a multiobjective optimization and commonality trade-offs[139]. They note that designers would benefit from having a visual analytics technique to help identify platform variables.

Khire uses multiobjective optimization coupled with additional visualization techniques[83]. This approach allows designers to iteratively and intuitively arrive at family platforms. Critical to this is their multiobjective optimization to find a band of solutions along the Pareto frontier. The band along the Pareto frontier is important because the performance penalty from component commonality prevents family products from being Pareto optimal. Including only the final converged front, limits the possible overlap that could be found. The Pareto bands are then displayed using various scatter plots. The decision maker can assess commonality by filtering points and inspecting the overlapping regions.

Slingerland proposes a hybrid approach for discovering appropriate commonality for a product family [151]. The study starts with the two family extremes where the components are all common and the other where all the components are unique. Then the parallel coordinate plots can show the trade-offs as commonality varies. One key benefit of using a parallel coordinate plot is that the bandwidths of different design variables become evident. Here, the bandwidth refers to the area that a particular design covers as it crosses an axis. If a particular product crosses at a wide section of the axis then it can indicate that the product is less sensitive to that design variable. Narrow bandwidths indicate a strong reliance on that design variable setting. As part of the Slingerland study they note difficulty in using parallel coordinates for discrete data as it is difficult to denote the actual proportions of points if they all overlay. To remedy this, Bendix generalized parallel coordinates to display multidimensional categorical data[11]. This modified presentation incorporates the frequency of individual data points by increasing the bin sizes and supports analysis of large and complex discrete data sets.

In addition to needing to display the multidimensional design variables of the family, it is important to be able to visualize similarity of a particular module space for which there are several options. The following approaches attempt to visualize the discrete partitioning decision variables that indicate family platforms. The main challenge of these techniques is to accurately capture the spectrum of partitions not just the all or nothing approach to sharing.

The first being dendrograms which visualize a hierarchy of nested partitions where the most similar items are grouped together first. Pedersen uses dendrograms showing the results of clustering together existing products to identify possible areas for future standardization [119]. Chen and Wang arrive at a dendrogram representation using a clustering analysis and Shannon's entropy theory on the multiple platform design problem[23]. Shannon entropy helps to select the platform variables once a clustering analysis is performed on individually designed products. Local clustering is performed for each platform to identify which subsets of products are more similar to each other than to others. Hölttä looks at using a dendrogram for the product architecting phase by creating a distance matrix between different components where the distance matrix was derived from functional requirements[68].

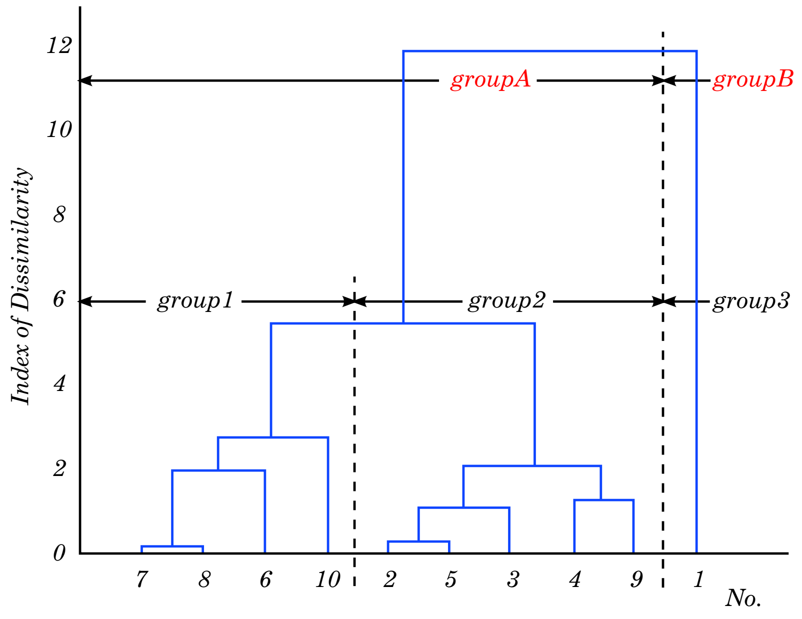


Figure 12: Dai Example of Sensitivity Clustering[33]

Dai and Scott[33] perform a hierarchical cluster analysis on design variables selected to become platforms based on sensitivity information from individually optimized products. Once the platforms are selected, the product variants are optimized to determine the values of the product specific design variables. If the performance loss due to the platform decisions is not acceptable, then a different product grouping is selected and the optimization process is repeated. Heuristics driven from the clustering results and trial-and-error were used to guide the platform configuration decisions leading to an optimized family. A dendrogram, figure 12, shows how the clustering results can be used to guide the selection of alternative partitions for the platform variables. For example, this dendrogram can be read by saying for an index of dissimilarity of 6 suggests three platforms be used. Furthermore, products $\{7,8,6,10\}$ should use one common component, products $\{2,5,3,4,9\}$ should use a different common component, and product 1 should use a completely unique component.

The primary benefit of using a dendrogram is that for a given level of similarity, there is a clear indication of the different partitions. Dendrograms can be limiting because they are only able to represent a hierarchy of nested partitions. This is undesirable because the best grouping may involve swapping one item from one group to another while changing the threshold, which could cause a large splitting of the partition. Fortunately, there are a few other visualization techniques to display

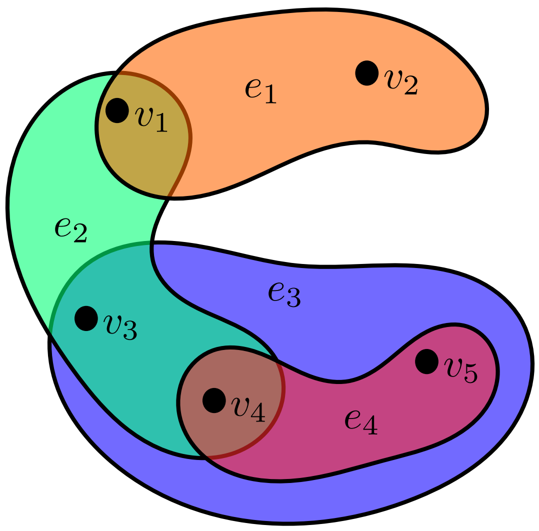


Figure 13: Example hypergraph

these possibilities. Hypergraphs, figure 13, can show families of sets and are a generalization of graphs in that an edge can include multiple vertices. In this case, a vertex is a product's component and vertices that are connected by an edge could be considered a platform. Obviously, to form a partition, several cuts through edges may need to be made.

Wei formalizes a hypergraph model for a product family's platform structure [167]. Unfortunately as the number of edges and products increase, the hypergraph may be complicated and difficult to generate[76]. To remedy this more traditional graph theory can be used with the edges weighted by the similarity between the different vertexes (components). Additionally heatmaps can also be used to show the pairwise strengths in a similarity matrix.

2.4 Gaps and Hypotheses

The literature review provides an understanding of the magnitude of the combinatorial problem faced in family design and shows that there are a variety of methods for trying to solve the multitude of issues in product family design. These methods can be categorized by the different classes of product family design problems they attempt to solve.

Class I problems are more traditional optimization problems because they do not include binary decision variables and solve only for the design parameters. This kind of problem arises when there is significant prior knowledge about the family. For example, in designing a new generation from products, it may be known which components worked well being shared or did not work well.

Making configuration decisions early helps simplify the product family problem by removing the combinatorial problem.

Class II problems involve only discrete decision variables that describe how components are shared across the different products. This kind of problem occurs when there already exists a library of components either from a previous generation of products or provided by suppliers. Because this class essentially needs to solve for the configuration of the family, the optimization is NP-hard[73].

Class III problems are even more difficult, because they require solving for the design parameters and the decision variables describing the platform configuration. In general the trend of research is toward solving the most difficult class of product families design problems. This class of problems requires identifying both the sharing of components across any subset of products in the family and all of the platform and unique design variables. Class III problems occur when there is little or no knowledge and many degrees of freedom to the design of the family. For example, this can happen when developing a new product family without the benefit of previous generations or when it is not immediately obvious which products would need similar components.

To make solving Class III problems simpler, most design methods will restrict commonality so that shared components are common across the entire set of products. Whereas, generalized commonality problems allow components to be shared across any subset of products but the combinatorial space quickly grows as the number of products being considered increases. The benefit of including generalized commonality is that it can yield a better performing family.

Methods for solving these classes of product family problems can be divided into two types: optimization and data mining. Optimization methods are critical to exactly specifying the ideal values of the design variables and the platform configuration of the family. However, because of the huge combinatorial space pure optimization approaches can be difficult. Frequently, this is done either in a multilevel optimization, or preferably, in a single stage if the problem is small enough [33]. Data mining methods using data either through a design space exploration or an optimization method are successful at identifying commonality in the family. More so, when a human is included during the exploration through proper visualization and understanding of the design space. However, many of these existing methods do not consider the complexity of SoS or are more qualitative in determining potential platform opportunities.

Many authors note the need for more product family methods to be able to deal with the high computational costs of Class III problems with generalized commonality. Moreover, none of the methods surveyed by the author include looking at a combined product family design problem that has system-of-systems elements. To help close the gap in assessing these complex system-of-systems product families, hypothesis 1 is put forth.

Hypothesis 1

Incorporating knowledge about potential family platforms into a larger product family design methodology can focus computational resources away from considering poor platforms making the design process more efficient and helping to identify the ideal tradeoff between product performances and portfolio costs.

This hypothesis needs to be further decomposed into more easily tested sub-hypotheses.

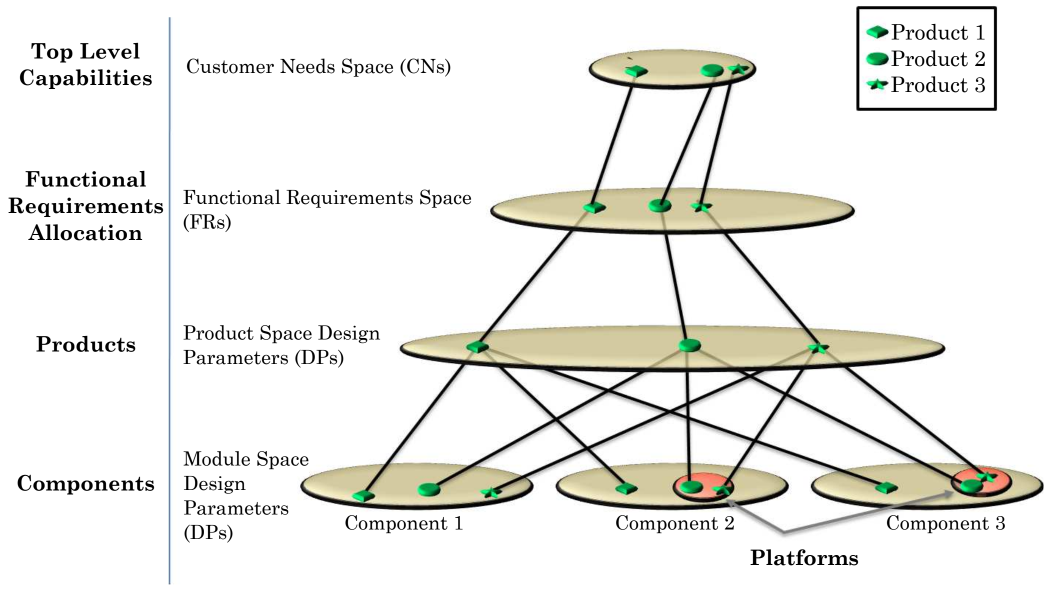
Sub-Hypothesis 1.1

If poor component combinations can be eliminated from consideration, then design resources can be focused on identifying the ideal trade-off between product performances and portfolio costs.

Sub-hypothesis 1.1 can be tested with a small thought experiment. The combinatorial problem is prohibitively large for a brute force approach to identify the best sharing of components. However, identifying the most dissimilar may be relatively easy. In an analogous way to a traditional screening test to remove unimportant variables, pruning obviously poor choices allows computational resources to concentrate in important regions. This then helps to identify the best trade-off between product performances and overall family costs. The other sub-hypothesis 1.2, relies specifically on observation 3.

Sub-Hypothesis 1.2

If combining components that are more similar makes a better family platform than combining dissimilar components, then a method to extract these patterns will aid in the formulation of the family.



Top Level Capabilities
Customer Needs Space (CNs)
Product 1
Product 2
Product 3
Functional Requirements Allocation
Functional Requirements Space (FRs)
Products
Product Space Design Parameters (DPs)
Components
Module Space Design Parameters (DPs)
Component 1
Component 2
Component 3
Platforms

Figure 14: Product Family Hierarchy

This sub-hypothesis then leads to an additional research question.

Research Question 2

How can family platform opportunities be systematically identified?

To help guide the solution formulation, we begin by investigating part of the breakdown of the different aspects in product family design. Figure 14 presents a hierarchy of domains based on the holistic view of figure 8. The top space of this hierarchy depicts the customer need niches that products must target to be successful. Below the customer needs space is the functional requirements space that each product must meet. The third layer is the product space where each product is defined by physical design parameters. The lowest level is the module spaces. Each module space is described by its own subset of design parameters from the product design parameters. Like in the holistic view, the mapping between the customer needs space and the functional requirements space is the product portfolio; mapping between the functional space and the product space is the product architecture; mapping between the module subspaces and the product space is the platform sharing configuration. There are three products represented in this hierarchy. The platforms can be seen in the Component 2 and Component 3 module spaces between Product 2 and Product 3.

This product family hierarchy is useful because it depicts the possibility of identifying family platform opportunities by recognizing patterns in the module spaces. Assuming a method for identifying commonality can be formulated, the next chapter proposes a generic product family methodology that provides context for extracting knowledge about alternative family platform configurations. This methodology addresses the gaps discussed above and is generic so that it can be used for any product family design problem, including those that begin with only the minimal amount of information, such as the customer's broad needs. Moreover, the family being considered can include products interacting at the system-of-systems level.

Once the methodology has been formulated, two different commonality identification approaches are formulated. Chapter 4 brings together a data mining clustering approach to recognize component commonality patterns. Chapter 5 utilizes a probabilistic graphical model as a basis for identifying patterns in component commonality.

Chapter III

DEVELOPMENT OF NEW PRODUCT FAMILY DESIGN METHODOLOGY

3.1 Family Design Methodology

The last chapter discusses the general types of family design problems, and establishes the need for a generic design methodology for product families with generalized commonality. There is an additional need for this methodology to address interactions between the products at a system-of-system level. This then extends the general Class III product family design problem to include family 'performance', $F(\bar{\mathbf{x}})$. So the goal becomes to maximize the family performance, the product performance, and the component commonality with respect to the design variables and the sharing decision variables subject to the product requirements and component commonality constraints.

$$
\begin{aligned}
&\text{maximize} \begin{cases} F(\bar{\mathbf{x}}) \\ f^p(\mathbf{x}^p) \\ \sum_{ijpq} R_{ij}^{pq} \end{cases} && \forall p \in \mathcal{P};\ p < q \\
&\text{with respect to} \begin{matrix} \bar{\mathbf{x}} = \left\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m\right\} \\ R \end{matrix} && m = |\mathcal{P}| \\
&\text{subject to } \mathbf{g}^p(\mathbf{x}^p) \le 0 \\
&\mathbf{h}^p(\mathbf{x}^p) = 0 \\
&R_{ij}^{pq}\left(x_i^p - x_j^q\right) = 0 \\
&R_{ij}^{pq} \in \{0,1\} \\
&(i,j) \in \mathbb{S}^{pq}
\end{aligned}
$$

where $\bar{\mathbf{x}}$ is the set of all the product design variables and $\mathbb{S}^{pq}$ is the design variables between product p and product q possible to be shared. The commonality relationships, R_{ij}^{pq}, are the source of the large combinatorial problem. The direct optimization approach is not feasible for these problems because they include generalized commonality and system-of-systems which greatly increase complexity. Generalized commonality increases product family design combinatorial space, while system-of-systems increase analysis model complexity. The complexity of these combined problems is difficult to overstate. In the generic SoS product family, trades occur between the products while maintaining the same customer need (CN). Different alternatives for accomplishing the same CN will have different family platform opportunities which may benefit more from commonality. The key element then is to understand the implications of commonality in the large SoS family problem, hypothesis 1.

Hypothesis 1

Incorporating knowledge about potential family platforms into a larger product family design methodology can focus computational resources away from considering poor platforms, making the design process more efficient and helping to identify the ideal trade-off between product performances and portfolio costs.

This methodology then needs to contain all of the steps and formalize alternative techniques necessary to system-of-systems family design starting with no or very little, knowledge. For a general framework, this product family design methodology uses the generic Integrated Product and Process Development, or IPPD, figure 15[134]. The IPPD methodology is formulated as an overarching umbrella covering the major development activities. The main set of decision making activities is down the center of the chart. This top-down design decision support process is facilitated with a computer integrated environment. The boxes to the left and right indicate possible tools that can be utilized for particular steps with arrows linking interactions and iterations between the different tools. The main iteration loop flows between generating and evaluating alternatives through the use of robust design and multidisciplinary design optimization. The methodology finishes with a selected design alternative that has proven capable of satisfying the problem objectives.

Using the generic IPPD methodology as a framework for a product family design methodology provides a robust framework for developing products. Similar to the IPPD, there is a main sequence. First the customer needs are established. Because this is also a SoS problem, details requirements for

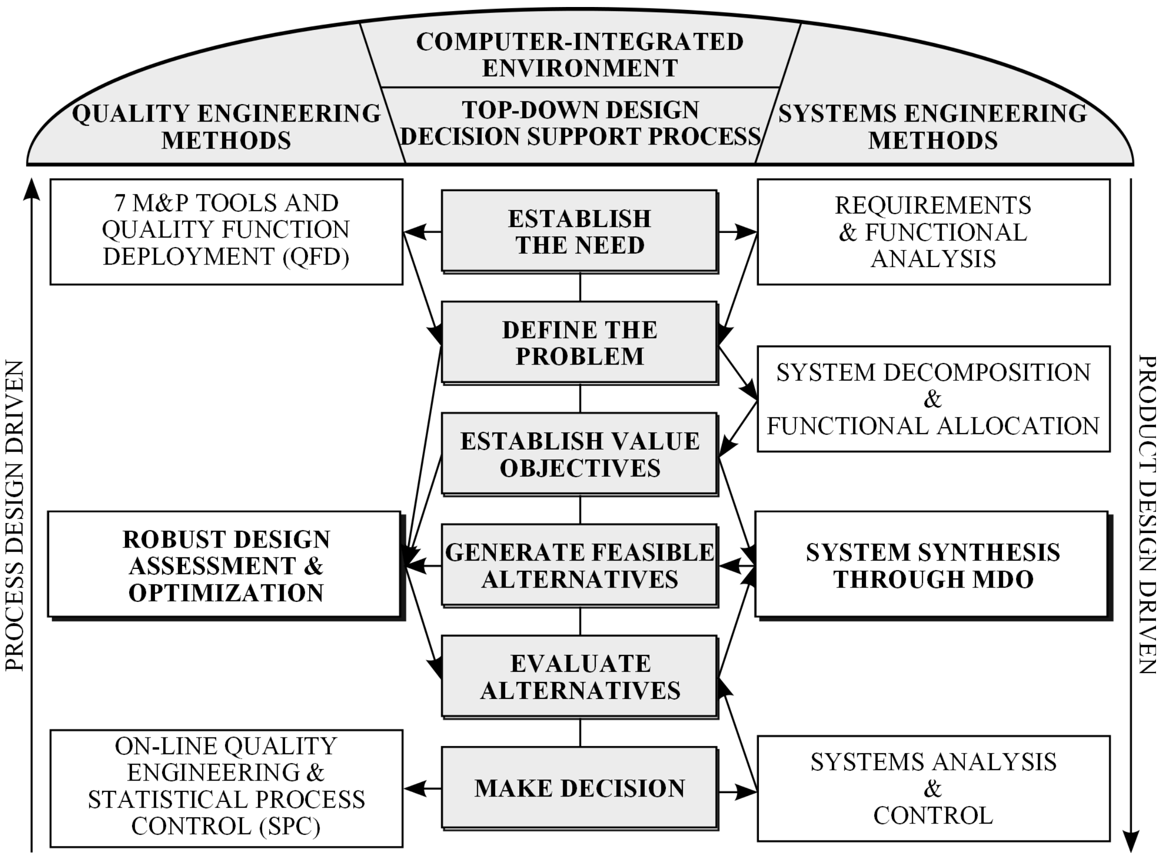

COMPUTER-INTEGRATED ENVIRONMENT
QUALITY ENGINEERING METHODS
TOP-DOWN DESIGN DECISION SUPPORT PROCESS
SYSTEMS ENGINEERING METHODS
7 M&P TOOLS AND QUALITY FUNCTION DEPLOYMENT (QFD)
ESTABLISH THE NEED
REQUIREMENTS & FUNCTIONAL ANALYSIS
DEFINE THE PROBLEM
SYSTEM DECOMPOSITION & FUNCTIONAL ALLOCATION
ESTABLISH VALUE OBJECTIVES
ROBUST DESIGN ASSESSMENT & OPTIMIZATION
GENERATE FEASIBLE ALTERNATIVES
SYSTEM SYNTHESIS THROUGH MDO
EVALUATE ALTERNATIVES
ON-LINE QUALITY ENGINEERING & STATISTICAL PROCESS CONTROL (SPC)
MAKE DECISION
SYSTEMS ANALYSIS & CONTROL
PROCESS DESIGN DRIVEN
PRODUCT DESIGN DRIVEN

Figure 15: The Georgia Tech IPPD Methodology

the different products are not known but different market niches can still be identified. Next, using the general CNs, designers can begin formulating alternative means to accomplish these needs. Once there is an idea of how the products operate and combine to meet the CNs then the general family value objects can be described. This also requires defining the module spaces and their parametrization through design variables.

For each product in the family, feasible alternatives are then generated using a modeling and simulation environment. Next platform opportunities are identified using a commonality identification approach. Through the course of this dissertation two approaches are developed to recognize the similarities between components. The first approach, chapter 4, is based on iteratively filtering and clustering the component subspaces to recover component partitions. The second approach, formulated in chapter 5, focuses on building a probabilistic model capable of encoding the joint distribution of the design space exploration and then using an inference process and earth movers distance to calculate similarity between component subspaces.

Once the platform alternatives are identified, additional trades can be performed to understand the product performance implications of the different possible family platforms. Finally, once the products and platforms have been identified and evaluated, the designer can make an informed family decision. As the steps are discussed in more detail below, connections will be made back to the formal problem description.

3.1.1 Step 1: Establish the Need

As with the product development process in figure 7, customer needs are critical and must be identified. If the niches are not well understood, the requirements allocated to the different products will not yield marketable results.

Part of this market understanding comes from examining the operating environment. For instance, a surveillance mission in mountainous terrain brings different functional requirements than a maritime surveillance mission. When predicting the product performance as it would be in the real world, the performance will have some variability. Analyzing the environment may reveal additional parameters that affect the stochasticity of the products' operating performance.

In addition to the original tools of the generic IPPD process, the family planner needs to be

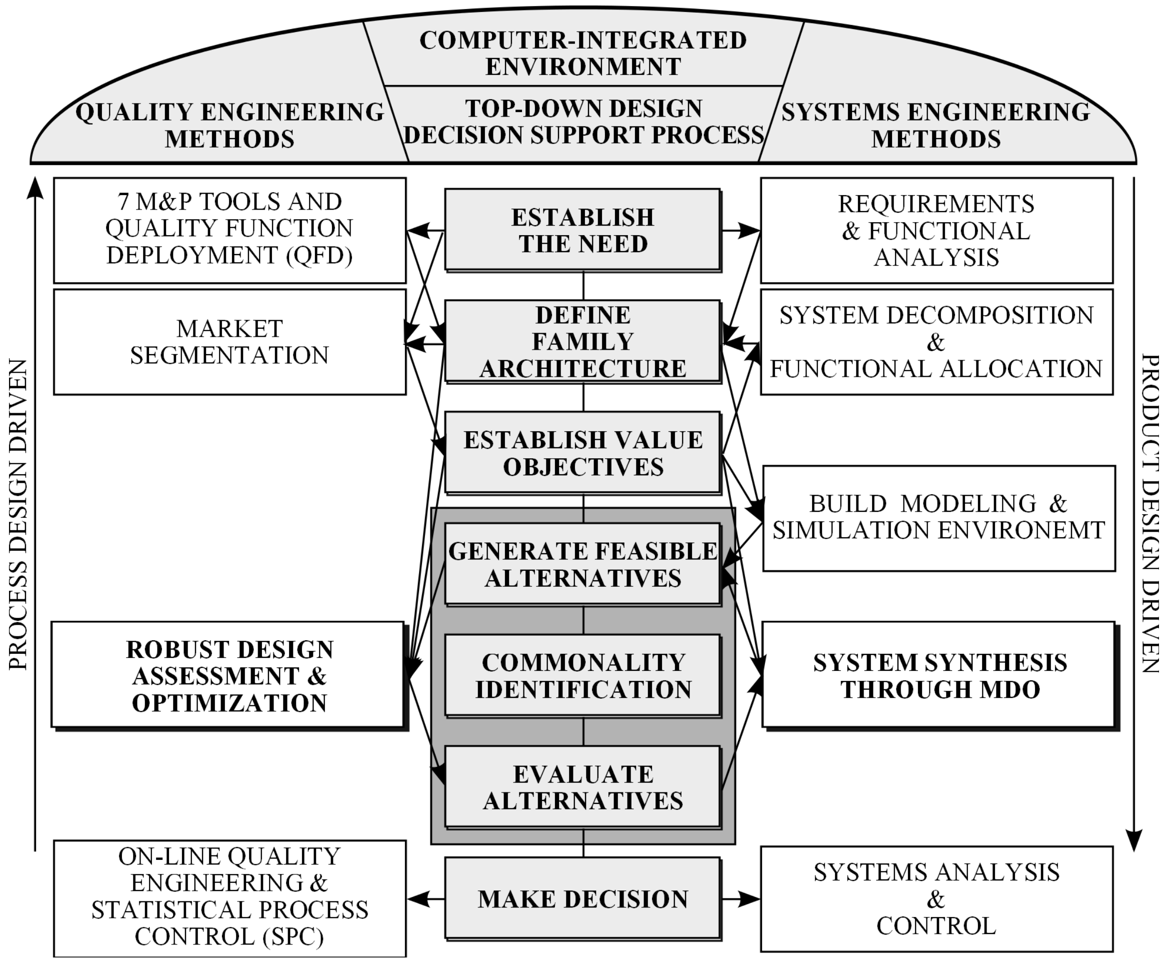

COMPUTER-INTEGRATED ENVIRONMENT
QUALITY ENGINEERING METHODS
TOP-DOWN DESIGN DECISION SUPPORT PROCESS
SYSTEMS ENGINEERING METHODS
7 M&P TOOLS AND QUALITY FUNCTION DEPLOYMENT (QFD)
ESTABLISH THE NEED
REQUIREMENTS & FUNCTIONAL ANALYSIS
MARKET SEGMENTATION
DEFINE FAMILY ARCHITECTURE
SYSTEM DECOMPOSITION & FUNCTIONAL ALLOCATION
ESTABLISH VALUE OBJECTIVES
BUILD MODELING & SIMULATION ENVIRONEMT
GENERATE FEASIBLE ALTERNATIVES
ROBUST DESIGN ASSESSMENT & OPTIMIZATION
COMMONALITY IDENTIFICATION
SYSTEM SYNTHESIS THROUGH MDO
EVALUATE ALTERNATIVES
ON-LINE QUALITY ENGINEERING & STATISTICAL PROCESS CONTROL (SPC)
MAKE DECISION
SYSTEMS ANALYSIS & CONTROL
PROCESS DESIGN DRIVEN
PRODUCT DESIGN DRIVEN

Figure 16: Generic Product Family Design Methodology using IPPD

able to segment the different customer needs either through extensive background research, market analysis or customer interactions. Ultimately the set of customer needs should be translatable into sets of Measures of Effectiveness (MoEs). For example, one customer may care about several related protection missions for the maritime personnel and environment, while a different customer may need to minimize damage from fires. These missions then can be described as sets of MoEs. These MoEs then serve as a basis for the SoS family evaluation $F(\bar{\mathbf{x}})$.

3.1.2 Step 2: Define Family Architectures

The goal of this step is to translate customer needs into engineering requirements and describe the set of functions each product needs to perform.

Family architecture can be defined as the allocation of functions (missions) to particular products and the corresponding product's modules[72]. The first step is to formulate and identify alternative functional or mission breakdowns for accomplishing the customer needs. In effect, the family architecture, therefore, describes the problem each product in the family needs to satisfy. Figure 17 shows three example architectures for a simple search and rescue mission. Architecture one uses an aircraft type for each mission, one for searching and a different kind of aircraft for rescuing. Architecture two uses one aircraft for both searching and rescuing. Finally, the architectures can also include different vehicle types like architecture three which has an aircraft searching and a helicopter rescuing.

To help guide the family architecture process, customer needs have been identified and, through market segmenting, grouped into target niches. With those target niches in mind, a system decomposition can be performed establishing the set of functions for the products. Once the functional breakdown is completed different family architectures can be developed. The family architecture relates the different functions back to physical systems. For complex systems there is no unique mapping between functions and systems resulting in additional trades to consider.

This is a complex step and may require iterations between functional allocation, defining the family architecture, the market segmentation, and the final objectives for each product.

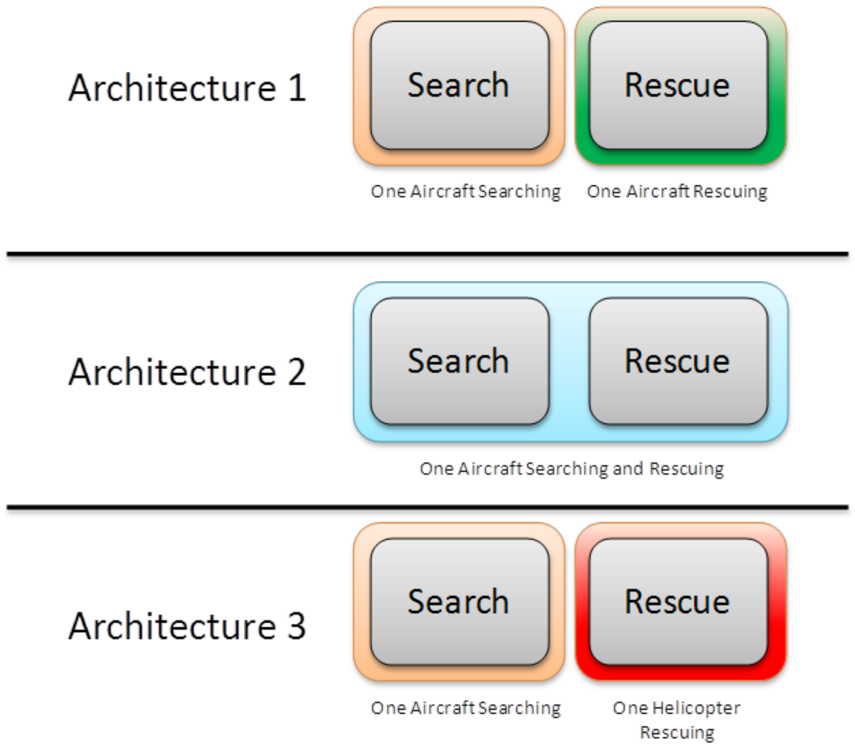


Figure 17: Satisfaction of Different Functional Requirements

The selection of the family architecture is not trivial and has important implications to the product and function breakdowns and components within each product. For instance, the set of components used in an aircraft are drastically different that the set of components used in a helicopter. Also alternative family architectures can have different numbers of products. All of these factors can significantly change the set of family platforms and sharing across the products.

3.1.3 Step 3: Establish Value Objectives

This step sets the requirements on both the products and the family. It attempts to answer which products are the most critical and which products and requirements are less important. These are relevant in determining the amount of acceptable trade-offs between the various products for the benefit of the family.

In addition to creating the function to system mappings, an individual product's modules should be identified, $\mathbb{S}^{pq}$. The module is a subsystem that can be treated as an individual product block and will be used later when looking for similarities between the different products in the family. One example is the determination of whether the family analysis considers the aircraft fuselage length and width to be a module or just the fuselage width. If the fuselage width is considered, then products may still differentiate their fuselages by adding plugs to extend them.

3.1.4 Step 4: Generate Feasible Alternatives

There needs to be a modeling and simulation environment to generate and evaluate product alternatives when there does not already exist any historical documents, like in the case of a new product family. The goal of the modeling and simulation environment is to calculate the impacts of low level component design parameters on high level customer needs. As an example, consider an airborne firefighting scenario where a customer is greatly concerned with the quantity of burnt land. In this scenario, the low level component design parameters may include fuselage length and width for the aircraft performing the mission, whereas the high level customer needs include the size of burnt land.

This new modeling and simulation environment is based off of models that adhere to the requirements placed on the systems found from the previous steps of identifying customer needs and family architecture, figure 18. Depending on the customer needs and how they are quantified, many different models may need to be created.

This modeling and simulation environment is typically decomposed into several different and more focused models. It can be easier to build these smaller models that have more well defined bounds. These smaller models output new parameters that act as inputs for higher level models. For example, a general vehicle performance model will output performance metrics like speed, fuel capacity, and fuel consumption. Those outputs are put into a SoS model to determine how effective the aircraft is in meeting the customer needs. Smaller models can also be more flexible, which allows them to be more easily reused in other modeling and simulation environments.

The models are typically physics-based and represent the physical processes involved. When creating the system models, care should be taken to address appropriate levels of fidelity. For instance, building a complex computational fluid dynamics model to predict the aerodynamic characteristics can yield more accuracy in the results. However, if the design space is too large, it is not feasible to perform the analysis. Typically, as physics based codes increase in fidelity, their computation time also increases greatly. One technique then is to build a surrogate model and replace the

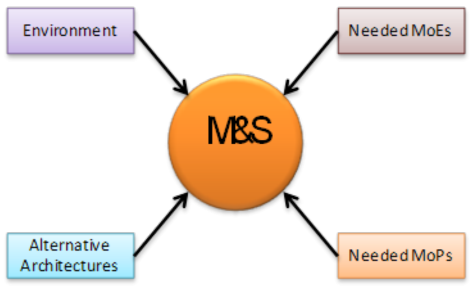


Figure 18: Needs of the Modeling and Simulation Environment

slower code. A surrogate model, like neural networks or response surface equations, are tuned to the underlying physics based models, capturing the relevant variability and can be executed in a fraction of the time. This speedup allows for a more densely explored design space useful for statistical analysis techniques. One difficulty in fitting surrogate models to complex and stochastic simulations is maintaining high prediction accuracy. If the physics are significantly complex, then the appropriate surrogate models may be as complex as the original model, preventing any evaluation speedups.

The end result of this step is to have a modeling and simulation environment that is able to capture the impacts of product components on high level customer MoEs. After the modeling environment is created, the product database is populated using a design space exploration technique like design of experiments (DoE), adaptive sampling, or optimization. This population of the database can be visualized on the product family domain hierarchy, figure 19. Here each dot would represent a product alternative.

The design space exploration depends on the class of the product family problem as well as any relevant engineering knowledge. Properly scoping the exploration through reasonable design variable ranges helps to improve the efficiency and accuracy of the results. If the family is a Class I problem, the platform configuration is known so the design space exploration can focus on generating design alternatives with the relevant component design variables linked across the products. In a Class II problem, the component design variables are known but the configuration is not. The exploration then needs to draw from these known component design variables when generating analysis cases. In Class III problems, both components and commonality are unknown making the exploration of the design space much more difficult. This research suggests exploring the design variables without component sharing and attempt to post process component similarity using one of

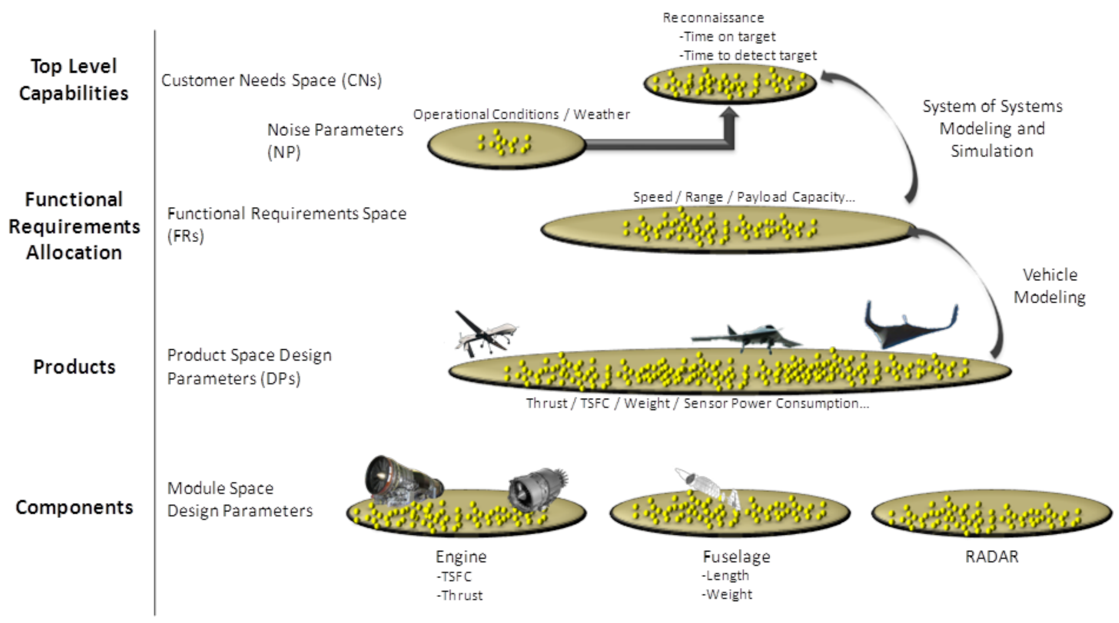

Top Level Capabilities
Customer Needs Space (CNs)
Reconnaissance
-Time on target
-Time to detect target
Operational Conditions / Weather
Noise Parameters (NP)
System of Systems Modeling and Simulation
Functional Requirements Allocation
Functional Requirements Space (FRs)
Speed / Range / Payload Capacity...
Vehicle Modeling
Products
Product Space Design Parameters (DPs)
Thrust / TSFC / Weight / Sensor Power Consumption...
Components
Module Space Design Parameters
Engine
-TSFC
-Thrust
Fuselage
-Length
-Weight
RADAR

Figure 19: Space Exploration

the commonality identification approaches.

This exploration step is critical because this generates the database of product design alternatives. If the database proves to be too sparse and fails to capture feasible points, then additional sampling schemes must be employed. Optimizers or adaptive samplers improve the database accuracy but these methods bias the design space exploration which may introduce artifacts in the database. Any patterns identified may be these artifacts instead of natural groups due to the physics and constraints of the family design problem.

3.1.5 Step 5: Commonality Identification

As stated previously, this methodology addresses Class III product family design problems with system-of-systems elements. So far, the steps in this methodology have been to build a simulation environment capable of exploring this complex family design space. Once the database is generated, there needs to be a technique to identify patterns in the module spaces. These patterns aid the platform selection by guiding detailed family exploration. Addressing this commonality identification need is research question 2.

Research Question 2

How can family platform opportunities be systematically identified?

The main goal for this step is to process the different module spaces and identify similarities between the products. It has been observed that if two components are more similar they will combine to form a better platform then if two dissimilar components are combined. To extract component similarity relationships, R_{ij}^{pq}, this dissertation formulates two possible approaches. The first is a data mining fuzzy c-means clustering approach, Chapter 4, applied on the individual module spaces. The second is a probabilistic approach, Chapter 5, and builds a probabilistic surrogate model of the design space exploration to then evaluate component similarities. The majority of the experiments, Chapter 7, performed for this dissertation are focused on testing these two commonality identification approaches and their effectiveness in extracting component similarity for use in this methodology. Also note, because components are similar does not necessarily mean they should be shared. Additional evaluations are needed to determine the performance penalty particular platforms introduce in the products.

3.1.5.1 Clustering Approach

The clustering approach is formulated and discussed in chapter 4 but will be included here for completeness.

Filter Design Space Once the database has been populated it can be post processed with different constraints to select the best design alternatives. These requirements can represent customer needs, feasibility constraints, viability constraints, and/or technical constraints, figure 20. Products that fail to meet their required customer needs are pruned and not considered in the pool of points to be clustered. The database is biased to the physics of the problem and filtering biases it to the problem constraints.

Identify Patterns in Component Design Spaces Once the database is filtered, the module spaces can be inspected to look for natural groupings in the design parameters. The goal is to be able to identify if any products are similar to one another for each module space. In figure 21, these notional

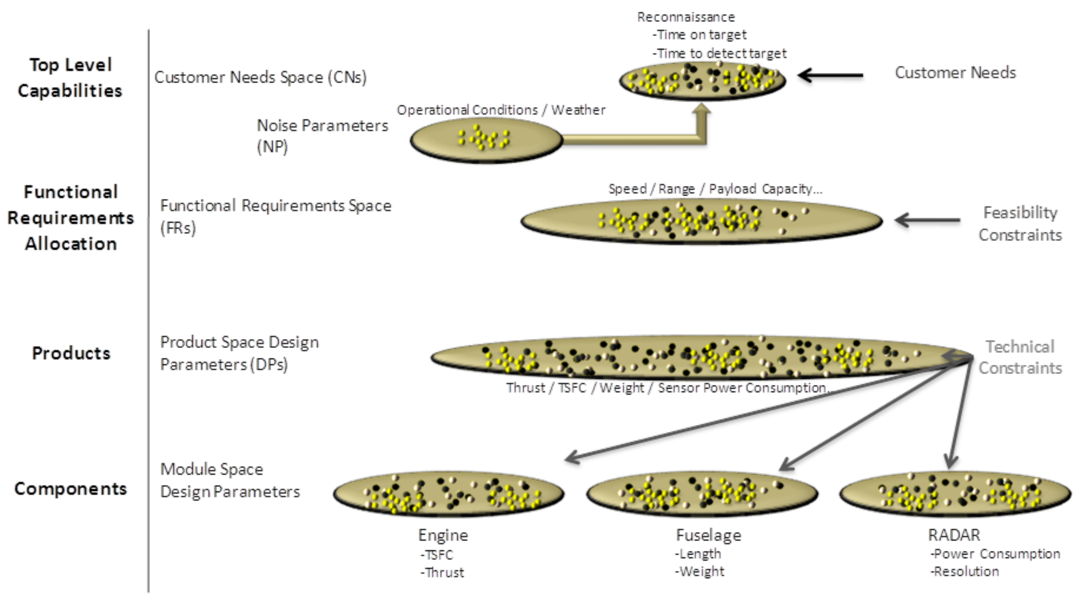

Top Level Capabilities
Customer Needs Space (CNs)
Reconnaissance
-Time on target
-Time to detect target
Customer Needs
Noise Parameters (NP)
Operational Conditions / Weather
Functional Requirements Allocation
Functional Requirements Space (FRs)
Speed / Range / Payload Capacity...
Feasibility Constraints
Products
Product Space Design Parameters (DPs)
Technical Constraints
Thrust / TSFC / Weight / Sensor Power Consumption...
Components
Module Space Design Parameters
Engine
-TSFC
-Thrust
Fuselage
-Length
-Weight
RADAR
-Power Consumption
-Resolution

Figure 20: Filtered Design Space

groupings would be between the middle product and the right product in the engine, fuselage, and radar spaces. Extracting these patterns can be done with a fuzzy c-means clustering for each module space. The fuzzy membership functions are then processed to yield component similarities between the different products.

Iterate The component sharing in one module space should affect the clustering and component sharing in a different module space. This interaction is captured by repeating the clustering with a new subset that modified the weights of the points with the cluster centroids for a given platform. The process converges once the weights of the subset do not change. The output of the process gives an indication of the platforms for each module and the centroid can give an estimate for the platform value.

3.1.5.2 Probabilistic Approach

This section gives an overview for the probabilistic approach to extracting component commonality but is formulated and discussed in detail in chapter 5. A Bayesian network model is first trained to a database and attempts to encode the joint probability distribution of the design space. Once the model is trained it can be used for a Bayesian inference given different requirements.

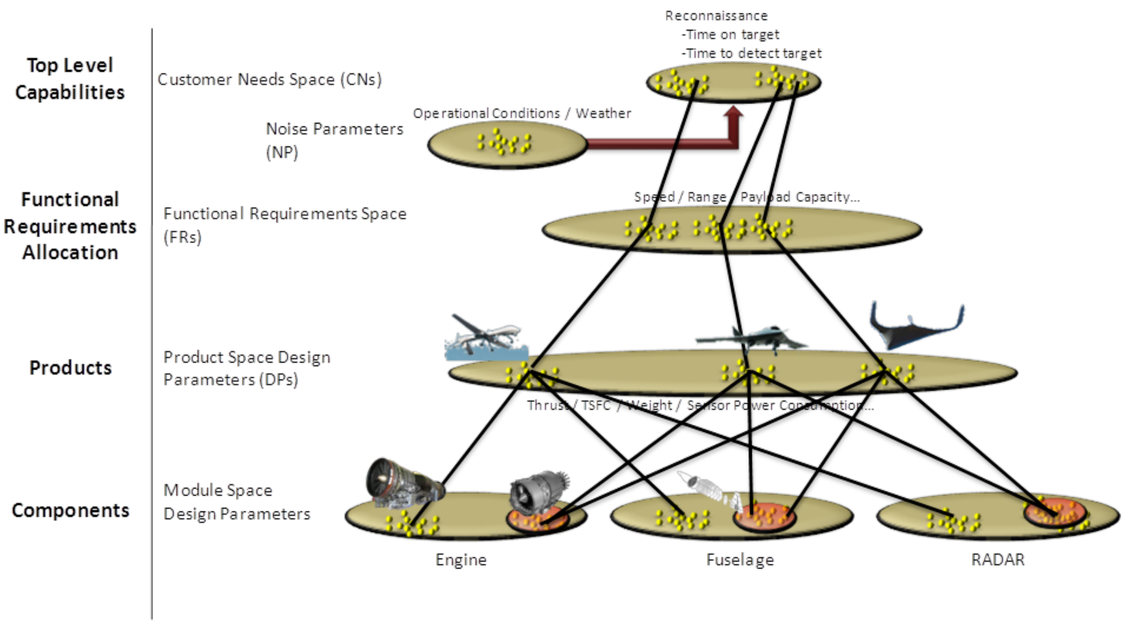

Top Level Capabilities
Customer Needs Space (CNs)
Reconnaissance
-Time on target
-Time to detect target
Noise Parameters (NP)
Operational Conditions / Weather
Functional Requirements Allocation
Functional Requirements Space (FRs)
Speed / Range / Payload Capacity...
Products
Product Space Design Parameters (DPs)
Thrust / TSFC / Weight / Sensor Power Consumption...
Components
Module Space Design Parameters
Engine
Fuselage
RADAR

Figure 21: Pattern Identification

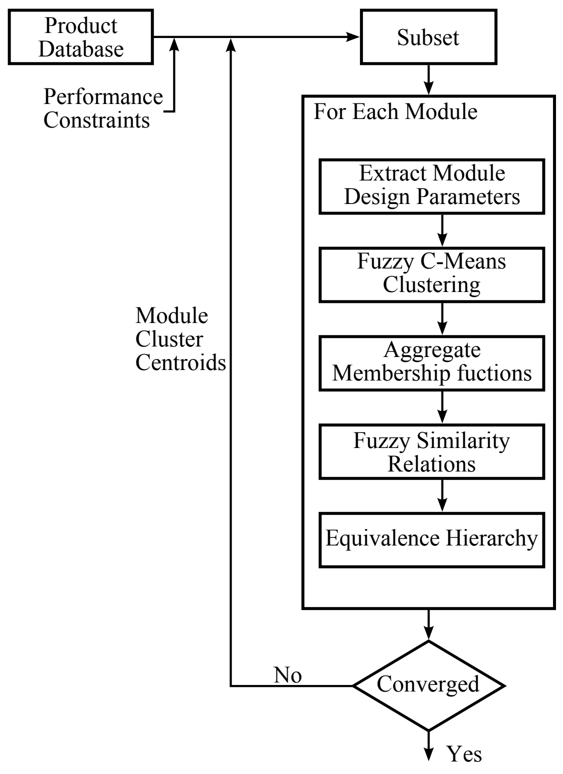


Figure 22: Platform Commonality Identification

Bayesian Network Training The Bayesian network again requires access to a database of design alternatives. The database can be generated using any design space exploration technique. Using the database, the network structure is learned to extract the relationships between the different problem variables. Structure learning is performed using a greedy search algorithm that calculates the Bayesian Information Criterion (BIC) score to help limit unimportant connections between variables. A metric like the BIC score requires the likelihood of the data for the given structure and conditional density so the conditional density of the nodes needs to be estimated. To help find a representative network structure, domain knowledge from subject experts can be included to prune bad edges or connect known variable interactions.

The network also needs to estimate the conditional distributions. There are several options but the one proposed for the experiments in this dissertation uses the Least Squares Conditional Density Estimation (LSCDE). It has been found to be a reasonably robust and quick method. There are two hyperparameters that are determined with a grid search to select the best score from a cross validation of the training data. The main purpose of building the Bayesian network is to perform inference and determine possible component commonality.

Commonality Reasoning Commonality reasoning attempts to determine similar components and likely platform values. This is done by performing importance sampling to infer the posterior distributions for the different design parameters.

These posterior distributions represent the likely values of those module design parameters that would give the product the desired performance. To calculate the module similarity, the earth movers distance is used calculate the pairwise distances between the posterior distributions. The result is a set of module similarity relations not unlike what the fuzzy similarity relations using the clustering approach.

The designer can then use these similarity relations to make informed decisions about the possible platforms in the family. The other benefit then is once the platform configuration is determined the platform posterior distribution can be calculated by combining the individual module posteriors that make up the partition. Finally this process of commonality reasoning can be iterated by repeating the Bayesian inference with product performance constraints and and platform distributions.

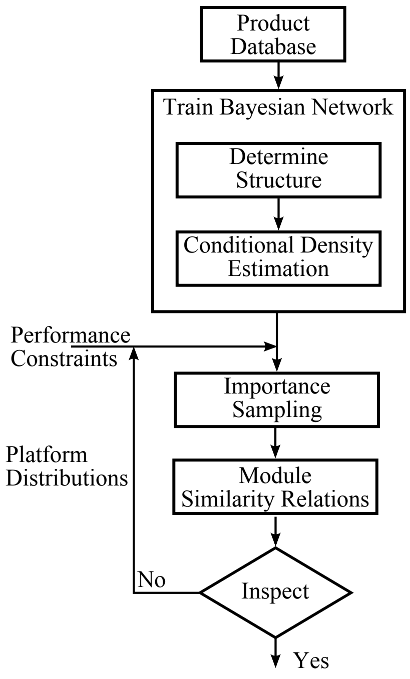


Figure 23: Bayesian Network Process

The posterior represents the distribution of that variable conditioned on the set of problem constraints. From the family perspective, if two products' components have a similar posterior distribution, then they would generate a better platform than if they were dissimilar. The earth movers distance is calculated between each product for each module space. These new module similarity relations ow can be visualized and inspected by a decision maker to select possible family platforms.

The outputs of the method are the inter-product similarities for the components. In addition, the results of the Bayesian inference enable some additional visualizations that can aid the designers in making informed commonality decisions. Finally with these decisions the impact of the platform configuration can be assessed and more easily traded against other alternatives with performance estimates obtained through subsequent inferences. Unlike the clustering approach, the probabilistic approach does not require successively filtering and reclustering subsets of the data as the variable interactions have already been encoded in the network structure and conditional densities.

3.1.6 Step 6: Evaluate Alternatives and Make Decision

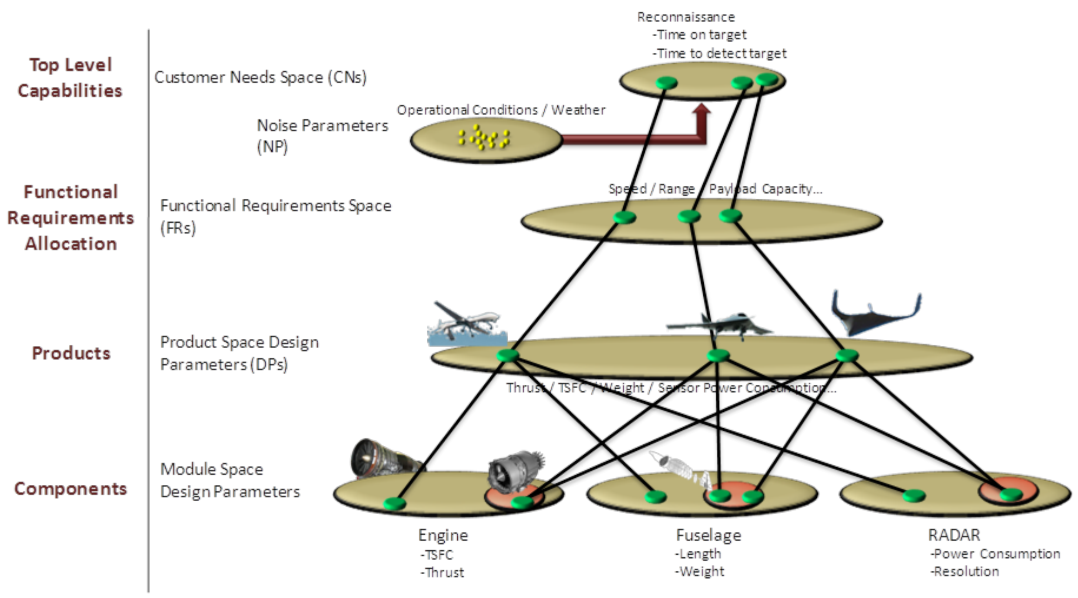
Assuming that potential platforms are identified, this step will take the alternative and proceed with higher fidelity models to increase the family planners knowledge. This knowledge enables the designer to make an informed decision and select the best family. Also now that platform similarities have been identified, the Class III problem has been reduced to trading several Class I problems against each other. The main benefit of this generic product family design methodology is that it takes knowledge extracted from the physics of the design space rather than knowledge from subject matter experts. Again, it was assumed in the beginning that there was not any knowledge about component similarities.

This step validates the family and architectural design space using different component sharing suggested by the preceding commonality detection. The end goal is to identify which architecture and family should be carried forward into more detailed design and analysis. Because the commonality decisions are guided by components that perform similarly, there should be less wasted computational effort on poor areas in the design space, which allows better searching in the preferred regions. From the notional examples above, sharing the engine, fuselage, or radar between aircraft one and aircraft two and three is not recommended. This is based off of clustering those module spaces and finding that product one was more different than product two and three.

It is also critical to evaluate the precise trade-off between cost and performance due to sharing in order to verify the suggested commonalities. The final objective is to arrive at a complete product family portfolio that maximizes the amount of cost savings while minimizing the performance penalties. Ultimately, this trade-off is what decision makers will use to select the family. Returning to the notional example of the aircraft, figure 24suggests that the trade-off in performance is not worth the cost savings for sharing a fuselage between aircraft two and three, but the performance penalty is not so great for the engine and radar components.

3.1.7 Remarks

This proposed generic product family methodology is needed because there is a gap in addressing complex system-of-systems product families where the component commonality is unknown. Because there is minimal knowledge about the product family provided, a sequence of generic steps are needed to formulate potential solutions. The crucial step in this methodology is the extraction of potential family platform configurations. Narrowing the number of alternatives for the family platforms, controls the combinatorial explosion of options and focuses computational resources into the



Top Level Capabilities
Functional Requirements Allocation
Products
Components
Customer Needs Space (CNs)
Noise Parameters (NP)
Functional Requirements Space (FRs)
Product Space Design Parameters (DPs)
Module Space Design Parameters
Reconnaissance
-Time on target
-Time to detect target
Operational Conditions / Weather
Speed / Range / Payload Capacity…
Thrust / TSFC / Weight / Sensor Power Consumption…
Engine
-TSFC
-Thrust
Fuselage
-Length
-Weight
RADAR
-Power Consumption
-Resolution

Figure 24: Evaluate Architectures

important regions of the design space. Although this methodology assumes limited prior knowledge, if additional knowledge were available, the problem becomes easier and the appropriate steps can be simplified.

Chapter IV

CLUSTERING APPROACH FORMULATION

This chapter reiterates the chief difficulty in product family design and develops a machine learning pattern recognition approach to identify commonalities between products in the family. From the fundamental product family trade-off, individual product performance decreases relative to their ideal as more components are shared. However, these shared components then reduce total family costs. If two optimized components are similar, then the performance penalty is smaller when making them a family platform. Therefore, if points are more similar to each other in a subspace then they will combine to make a better platform than if they were to form a platform with points that are more dissimilar.

Inspecting the hierarchy of product family domains, illustrated in figure 14, we can detect some similarities in the module subspace. Two products appear that have very similar component 3s and components 2s. This similarity may then be a useful commonality heuristic to further exploration of the product family. Human beings are very good at identifying patterns but are limited to smaller and lower dimensional problems. A method to extract these patterns will aid in the formulation of the family. Formally, pattern recognition is a process to analyze and extract possible similar features from data and is a significant role in many machine learning problems[141].

Machine learning is a broad field in computer science that specializes in algorithms that can automatically detect patterns in data[106]. The goal then is for these extracted patterns to be used to make predictions. The set of methods that machine learning can be organized into supervised learning, unsupervised learning, and reinforcement learning. In supervised learning algorithms takes a training set of labeled data and creates a classifier that can then predict the correct label for any new data. Unsupervised learning attempts to discover natural patterns inside the unlabeled data. Reinforcement learning algorithms are useful for applications that require rewarding positive behavior and penalizing negative behavior. Identifying commonality between components of the family is not a task with known labels or suited to rewarding behavior, which leaves unsupervised

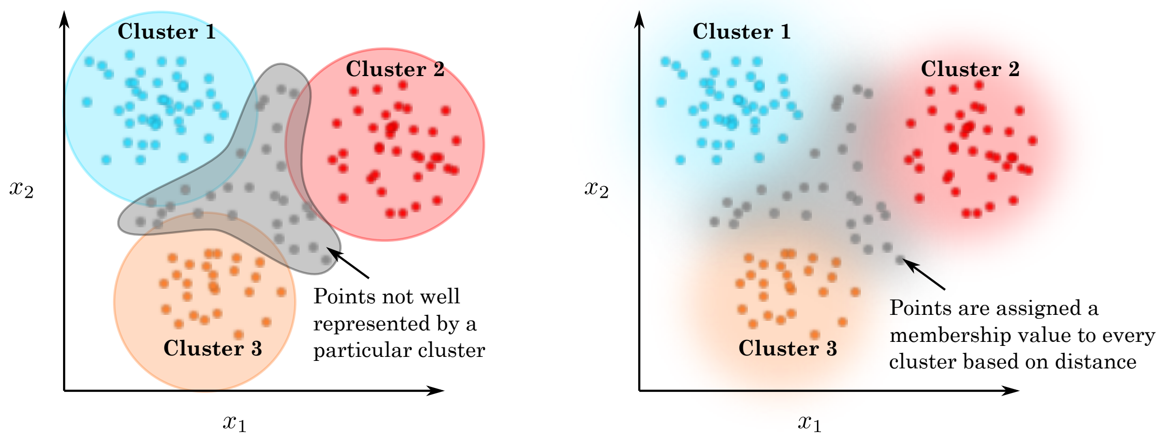

Cluster 1
Cluster 2
Cluster 3
x_2
x_1
Points not well represented by a particular cluster
Cluster 1
Cluster 2
Cluster 3
x_2
x_1
Points are assigned a membership value to every cluster based on distance

Figure 25: Comparison of Traditional Clustering (left) Versus Fuzzy Clustering (right)

learning.

Inside the branch of unsupervised learning, there are many methods for extracting structure from the data but can be divided into dimensionality reduction and clustering methods. Dimensionality reduction methods attempt to collapse the problem onto the most relevant subspace and include principal component analysis. Clustering methods sets out to divide data set so that the most similar points are in the same group (cluster) and will serve as a starting point for the product family component data mining approach.

4.1 Cluster Analysis

Clustering analysis is a "collective term covering a wide variety of techniques for delineating natural groups [7]." There are two branches of clustering methods, hard and fuzzy clustering. In hard clustering each point in the data set is clearly assigned to one cluster or another while in fuzzy clustering each point is given a likelihood to all the clusters [159]. This likelihood is called a membership function but can be thought of as the probability that point belongs to a particular cluster. Figure 25 shows a comparison between traditional crisp clustering and fuzzy clustering.

In the domain of product family design it is unlikely that the database is dense. This means that the points are only near good platform settings and not exactly on them. Furthermore, in a generic product family design problem the product may operate in a noisy environment. Consequently, the points would then be further distributed around the idea platform settings. Because the data is less crisp and imperfect, fuzzy clustering would be more robust in identifying points that are not strongly represented by any particular cluster. This is a good property for a starting point in developing the background for a pattern recognition product family design methodology.

Sub-Hypothesis 2.1

If a sufficiently accurate and dense database can be generated, then a machine learning pattern recognition technique like fuzzy clustering could be used to help identify component commonality and potentially form a platform.

4.1.1 Fuzzy Clustering

Fuzzy clustering is beneficial for trying to identify relationships in noisy data sets because it captures the fuzziness that exists in the clustering process and design space in the membership function. The membership function, $U = \left\{u_{ij} | \{i,j\} \in c \times n\right\}$, encodes the fuzziness of the problem where c is the number of clusters specified a priori, and n is the number of design points being clustered.

Fuzzy clustering can be further broken down into different algorithms each with different variations on how to perform the update procedure and how to calculate the cluster centroids and slightly modified objective functions. In Fuzzy C-Means (FCM), clustering the objective function J, to minimize is[36]

$$J\left(\mathfrak{M}_{\mathfrak{l}}, U, V\right) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} \tag{3}$$

Where m is a fuzziness parameter, as m increases, the boundaries between clusters becomes more fuzzy; d_{ij} is a similarity measure from the j^{th} point to the i^{th} cluster's centroid. There are several different types of similarity measures that could be used to calculate the strength of the closeness [77]. Santini summarized similarity measures (or dissimilarity) to a distance in some metric space[133]. There are four properties necessary for a similarity measure d. First there is self similarity:

$$d\left(A, A\right) = d\left(B, B\right)$$

Minimality:

$$d(A,B) \geq d(A,A)$$

Third symmetric:

$$d(A,B) = d(B,A)$$

Triangular inequality:

$$d(A,B) + d(B,C) \geq d(A,C)$$

It is too long to list all similarity measures but the most common tend to be: euclidean distance, Manhattan distance, and cosine distance. For each iteration, the membership function U is updated using equ 4, and cluster centroids $V = \{v_1, \ldots, v_{i,} \ldots, v_c\}$ are recalculated using Equation 5 [36]

$$u_{ij} = \frac{d_{ij}^{-\frac{2}{m-1}}}{\sum_{i=1}^{c} d_{ij}^{-\frac{2}{m-1}}} \tag{4}$$

$$v_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}} \tag{5}$$

where x_j is the j^{th} point's design parameters representing module $\mathfrak{M}_{\mathfrak{l}}$. During the update procedure, two constraints are applied, Equation 6 and 7

$$\sum_{j=1}^{n} u_{ij} > 0, \forall i \in \{1, \ldots, c\} \tag{6}$$

$$\sum_{i=1}^{c} u_{ij} = 1, \forall j \in \{1, \ldots, n\} \tag{7}$$

Together, they ensure that no cluster is empty, and that the sum of the membership of any data point to each cluster is one. Because the memberships are $[0,1]$ and their sum is one, the membership function can be interpreted as the probability of that data point belongs to that particular cluster.

In FCM, the number of clusters is specified a priori and should be varied to find the highest compactness with the furthest cluster separation. Results of the clustering can be tested using validity indices which measure the compactness and separation. Compactness is a measure of the variation the data has within each cluster, and separation measures inter-cluster similarity[36]. One such validity index is the Xie and Beni index [36].

$$XB(c) = \frac{\sum_{i=1}^{c}\sum_{j=1}^{n}(u_{ij})^{m}||x_j - v_i||^2}{n \cdot \min_{i,j}||v_j - v_i||^2} \tag{8}$$

The optimal number of clusters is the one where the XB index is the lowest and is the one whose membership functions are used. Effectively the membership functions relate each data point to each cluster. However for this to apply to family commonality both points and clusters are only an intermediate step. Points from the database are only possible designs from a design space exploration method but are still representative of a particular product with that products performance. The clusters are an intermediate container for then processing the membership functions to extract the similarity between the modules.

4.1.2 Module Similarity

After clustering, the membership functions relate each data point to each cluster but not the global similarities of one module to other. In other words after the clustering, we know in which clusters products appear but not the similarity from one product to another product.

To identify similarities across different products, more processing is necessary. The following sets up the theory for extracting binary similarity relations from the bulk membership functions. These steps are similar to a market segmentation method for product family positioning based on fuzzy clustering is presented by Zhang [170].

Given the membership functions an average membership function is agglomerated by product. The average membership function $U^* = \left\{u^*_{ip}|\{i,p\} \in c \times F\right\}$ describes the likelihood product p belongs to cluster j is equ 9 [135]

$$u^*_{ip} = \frac{1}{|F_p|}\sum_{j}^{|F_p|} w_j u_{ij} \, \forall j \in F_p \tag{9}$$

where w_j is the weight of the design alternative and j would be all of the rows in the database that represent product p. The weighting is there so that points that are better performing design alternatives have a stronger weight, and affect the total product's membership function more than points that perform poorly.

After finding the average membership function based on natural grouping in the database, the next step is to identify the similarity between product components. The product-to-cluster membership matrix is then used to generate a binary similarity relation between all the products. Fuzzy similarity relations can be used to quantify these similarities. Fuzzy binary relations, $R = \left\{ r_{ij} \, \forall \{i,j\} \in F \times F \right\}$ can be generated through several different fuzzy operations. The most restrictive fuzzy operation involves summing the minimum between the average membership function of two different products across all of the clusters equ 10 [48, 13].

$$r_{ij} = \sum_{z \in c} \min\left(u^*_{zi}, u^*_{zj}\right) \tag{10}$$

In effect, if two products appear in the same clusters, then they are more similar to each other than to a third product that does not appear in the same clusters. There are two main properties of fuzzy similarity relations, equ 11-12 [77].

Reflexive:

$$R_{ii} = 1, \forall i \in F \tag{11}$$

Symmetric:

$$R_{ij} = R_{ji}, \forall \{i,j\} \in F \times F \tag{12}$$

This summing operation yields numbers that are between zero and one with one being identical and zero being independent. However the binary similarity relationship is not transitive. Transitivity is a useful property because the relationship product one and product two share and the relationship product two and product three share, is the same as the relationship between product one and product three. By making them transitive a fuzzy equivalence relations is generated. Being transitive also allows the visualization of the fuzzy equivalence relation with a dendrogram.

A transitive closure is accomplished by equ 13.

$$R^* \subseteq R \circ R ... R \tag{13}$$

Where the composition $R \circ R$

Table 2: Example Fuzzy Equivalence Relation

	1	**2**	**3**	**4**
1	1	.2	.6	.2
2	.2	1	.2	.8
3	.6	.2	1	.2
4	.2	.8	.2	1

$$R \circ R = \max_j \left(R_{ij}\, t\, R_{jk}\right) \forall \{i,j,k\} \in F \times F \times F \tag{14}$$

The process for creating the fuzzy equivalence is by repeatedly taking the $R \circ R$ composition, equ 14, until the resulting relation converges with the input relation. In this equation t represents a (triangular-norm) or t-norm. A t-norm is a common operation in fuzzy logic and is the fuzzy equivalent of the triangular equality. One of the most restrictive t-norms is the max-min equ 15.

$$R_{ik} \geq \max_j \left\{\min\left[R_{ij}, R_{jk}\right]\right\}, \forall \{i,j,k\} \in F \times F \times F \tag{15}$$

Fuzzy equivalence relations $R^* = \left\{r^*_{ij} \forall \{i,j\} \in F \times F\right\}$, are still $r^*_{ij} \rightarrow [0,1]$, reflexive, and symmetric. The fuzzy equivalence relations can be made crisp by assigning elements equal to one if they are above a threshold. This α-cut similarity threshold is given by equ 16.

$$r_{ij}(\alpha) = \begin{cases} 1 & r^*_{ij} \geq \alpha \\ 0 & r^*_{ij} < \alpha \end{cases} \tag{16}$$

Finally, by inspecting the elements of $R^*(\alpha)$, the equivalence classes representing unique partitions are extracted. Equivalence classes are subsets of F that are all like elements.

$$a = \{b \in F / R^*(\alpha) = 1, \alpha \in [0,1]\} \tag{17}$$

By taking different α-cuts different equivalence classes are created from the fuzzy equivalence relations. These different equivalence classes form a nested hierarchy of partitions of the different product modules. If $\alpha > \beta$, then $R^*(\alpha)$ yields a finer partitioning than $R^*(\beta)$.

As an example, consider the fuzzy equivalence relation in table 2. Products two and four are the most similar followed next by one and three.

Table 3: Example Fuzzy Equivalence Relation α-cut of .8

	1	**2**	**3**	**4**
1	1	0	0	0
2	0	1	0	1
3	0	0	1	0
4	0	1	0	1

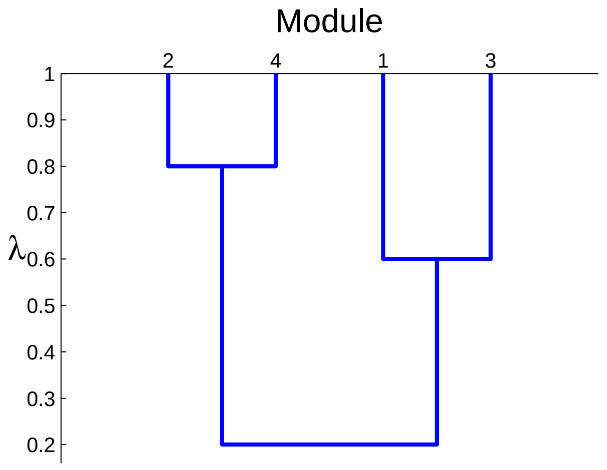


Figure 26: Example Equivalence Class (sharing partition) Dendrogram

Table 3 shows an α-cuts of $\alpha = 0.8$. Values in the original fuzzy equivalence relation below 0.8 have been zeroed.

We can identify the nested sharing hierarchy represented by this fuzzy equivalence relation graphically by taking the resulting equivalent classes using cuts at $\alpha = \{0, 0.2, 0.6, 0.8, 1\}$, figure 26. The y axis of the dendrogram shows the partitions for each α cut. At high, $\alpha = 1$, all of the products are independent and as α decreases the platforms are created. This can be used in a similar way to figure 12 [33]. Product 1 and 3 are more similar to each other than product 2 and 4. As a heuristic this can guide further exploration of the family exploration process to at least not include certain combinations.

4.2 Design Space Exploration

One critical aspect of this data mining approach to identify product family commonality that has not already been discussed is the origin of the data. Because this is for a generic product family design problem there may not be any historical database to mine. However through the use of modeling and simulation, the impacts of the design parameters can be seen on the relevant output performance metrics. From a philosophical standpoint, how the input design parameter combinations are generated is important. The entire purpose of this approach is to identify patterns, but what if the patterns originate from the method used to explore the design space? The exploration should be unbiased to ensure the patterns result from the physics of the design problem and the customer requirements rather than artifacts from the exploration. To understand the impact of the database on the formulated clustering approach, the experiment two will use different databases generated from alternate methods. These methods are a Design of Experiments (DoE), Monte Carlo simulation, and a multiobjective evolutionary optimizer.

Design of experiments are defined as "a systematic, rigorous approach to engineering problem-solving that applies principles and techniques at the data collection stage so as to ensure the generation of valid, defensible, and supportable engineering conclusions [111]." There have been numerous different kinds of design of experiments generated each with different pros and cons. They all attempt to generate the combination of cases to be analyzed in such a way as to try and maximize the information gathered.

In design space exploration, space filling DoEs are most commonly used. Of these are full factorial, Latin Hypercubes (LHC), sphere packing, and Hammersley sequence. Recently it has been found that the Hammersley sequence provides a more uniform coverage[146] than Latin Hypercube designs which are the most common.

Monte Carlo is a design of experiments where the cases are purely random generated by treating the input parameters as random variables with particular distributions (usually uniform between lower and upper bounds). In Monte Carlo simulation the input samples are generated quickly so large numbers of points can be generated. Biltgen demonstrates how Monte Carlo simulation combined with fast models allow for bulk data examination [14].

Exploring the design space with either a DoE or a Monte Carlo simulation will not likely be focused near the Pareto frontier and will include many dominated solutions. Unless there is a huge family cost savings for having common components that compromise the product's performance, the products still will remain near their Pareto optimal. So if the DoE and MC databases are low in sample size then they will have poor coverage near the frontier and be of low quality. Which means that any patterns identified could be far from the ideal family commonality.

To that end a multiobjective evolutionary algorithms (MOEAs) can be useful for generating a database with better quality. The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is one such algorithm for identifying a set of non-domination solutions along the Pareto frontier unlike traditional optimizers that only return a point solution based on a single objective function [37]. An example of a Pareto frontier is shown in figure 27. This set represents the best possible designs which means that to improve in one objective requires a trade-off in another.

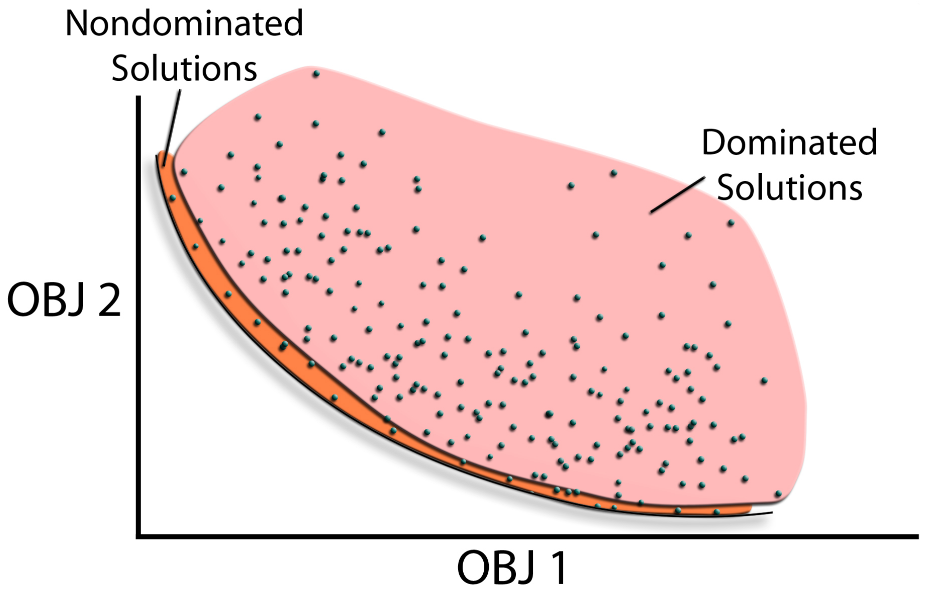


Figure 27: Pareto Frontier Example

The key to the NSGA-II is the dominance sorting. If a member in the population is better than another member in all objective dimensions it is said to dominate the other member. The algorithm for the NSGA-II dominance sorting is as follows [37]. For each point in the population compare it to every other point. If p dominates q then add q to the set of points dominated by p. However, if q dominates p then increment the domination counter of p. Once all of the population has been compared, if the domination counter of p is zero then it can be added p to the first front.

Algorithm 1 NSGA-II Dominance Sorting

```
for each $p \varepsilon P$ do
    $S_p = \emptyset$
    $n_p = 0$
    for each $q \varepsilon P$ do
        if $p \prec q$ then
            $S_p = S_p \bigcup \{q\}$
        else if $q \prec p$ then
            $n_p = n_p + 1$
        end if
    end for
    if $n_p = 0$ then
        $p_{rank} = 1$
        $Front_1 = Front_1 \bigcup \{p\}$
    end if
end for
```

After the dominance sorting is done and the first front is found, the next front is found by looking at the set of points dominated by the points in the first front. If no other points dominate a member other than those belonging to the first front, then that point belongs to the next front. This process then continues until all of the points are assigned into a front. Like traditional genetic algorithms, the NSGA-II uses a mating mechanism where mating preference is given to the less dominated members of lower fronts. Once the new population has been generated and evaluated, both the original mating population and the new population are sorted based on dominance and only the lower fronts are kept. Eventually the lowest front will converge to the Pareto frontier. The NSGA-II can also be adapted for use with constraints. This is done by expanding the definition of the non-dominated sorting. When taking into account constraints, dominance sorting proceeds as before but infeasible points are automatically dominated by the feasible set of points [37].

Although the NSGA-II will find the Pareto frontier the points were generated in a guided manor and are not uniform. Recall then that any patterns found could then be the result of the exploration method and may suffer accordingly.

4.3 Clustering Approach

This chapter began with a study into pattern recognition methods to extract component relationships in a product family. From these methods, a fuzzy c-means clustering approach maintains a finer level of information about the data using membership functions. Fuzzy operations can then process the membership functions to generate a hierarchy of partitions similar to the dendrogram in figure 26.

A product database needs to be generated so it can be mined for information. In the previous section there are several methods to explore the design space and to and populate the initial design database. After the database is populated, the module spaces can be extracted to look for natural groupings in the design parameters. Knowing which design parameters represent each module space is part of the product architecture process. Each of the design parameters should be normalized to prevent dimensions of large magnitude from skewing the clustering results. Then a fuzzy clustering algorithm can be used on each component space with care being taken in determining the number of clusters used.

At this stage the clustering in one module space should affect the clustering in a different module space. This interaction is captured by repeating the clustering with a new subset that modified the weights of the points with the cluster centroids for a given partition. The process converges once the weightings of the subset does not change. The output of the process gives an indication of the platforms for each module and the centroid can give an estimate for the platform value.

Now that the clustering approach has been formulated, recall one of the key observations about the motivating aerospace product families. Observation 1 finds aerospace products with their long life-cycles especially susceptible to the need to adapt to future requirements. This fundamental need to incorporate flexibility into a product family can be captured if the component sharing sensitivity can be calculated.

Sub-Hypothesis 3.1

If the design constraints are changed then using the new feasible subset from the design database and performing the pattern recognition will reveal the sensitivity of component sharing

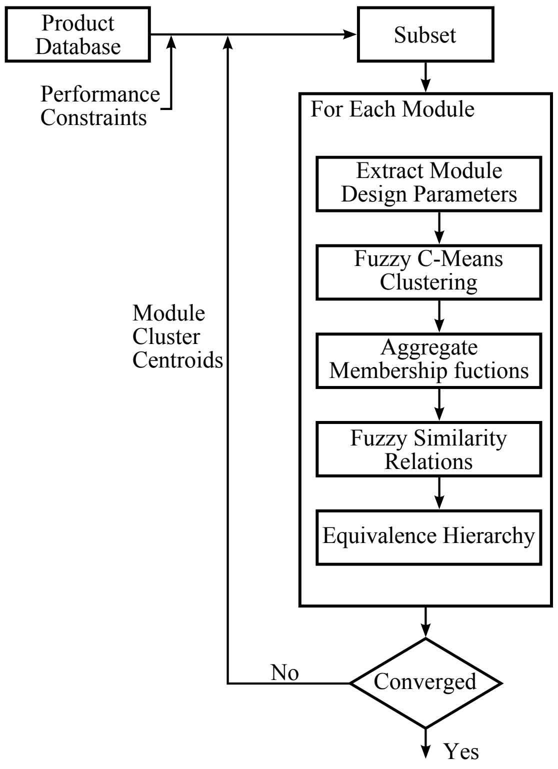


Figure 28: Platform Commonality Identification

There are three experiments using an electric motor family design as a benchmark. The first experiment is to test the validity of this clustering approach. It is noted that this clustering approach relays heavily on the quality of the design database to be able to accurately determine the platforms. This experiment included a second refined exploration using narrowed design variable ranged found from a courser Latin Hypercube DoE.

Experiment two is formulated to test which method of exploration is most efficient and how sensitive the clustering approach is to the initial size of the database. In this experiment it is believed that the predicted error for the platforms predicted from the clustering will decrease as the population size of the design database increases. This is because there should be more points around the family solution in the benchmark study. Furthermore, the ability of an evolutionary algorithm is also included in this convergence study. It is postulated that an evolutionary algorithm may introduce bias in the design database skewing the results of the clustering.

The third experiment uses a different benchmark study to understand how changing the performance requirements changes the commonality relations.

Finally it may not be possible to generate a design database that has enough resolution to accurately predict the platform variables. This is especially possible for more a complex family where the analysis codes have longer run times. The following chapter 5 introduces a competing method for extracting commonality using a probabilistic surrogate approach to detect patterns. It is hoped that build a representation of the database which can be interrogated more quickly will scale to more difficult problems better.

Chapter V

PROBABILISTIC APPROACH FORMULATION

Probabilities play a central role in modern pattern recognition[17], and this chapter builds a novel probabilistic approach to the family platform configuration problem. This probabilistic approach attempts to build a surrogate model of the design space exploration by using a Bayesian network to capture the joint probability distribution. Training a Bayesian network requires learning the network structure and then estimating the conditional distributions of the network nodes given their parents.

After training, Product performance requirements can then be applied as evidence to the network. Using statistical inference techniques, this evidence propagates through the network and yields the posterior distributions for the component design variable nodes. In effect, these posterior distributions indicate the likely values for the design variables given the requirements.

Identifying product component commonality relies on comparing the similarity between the conditional distributions of component design variables. If two products' components have a similar distribution, then they could be combined to become a family platform. The platform design variable distribution can then be estimated by combining the component design variable distributions from the relevant products.

5.1 Surrogate Modeling

For a generic system-of-systems product family design problem there are very many dimensions and because of the SoS nature possibly stochastic. This adds many restrictions onto the kinds of acceptable surrogate models for representing the point cloud of the product family design space exploration.

A model is an abstraction of reality and is not a perfect representation of that reality[61]. Whenever a model is created, only some aspects of reality can be incorporated and others must be removed. Part of the science of surrogate modeling is understanding which dimensions have the most impact. Imagine reality being a truth model where some function f maps the domain x into the range y

$$y = f(x)$$

Because this function is unknown a surrogate model would find some estimate, $\hat{y}$, of y being

$$\begin{aligned} y &= \hat{y} + \varepsilon \\ \hat{y} &= g(x) \end{aligned}$$

where ε represents approximation error and measurement (random) errors and g is is the model[145]. Because the surrogate is usually significantly faster to execute than the original function, it can be used to understand the relationship between x and y. In creating a surrogate model there are four steps:

1. Experimental design - generates the combinations of input parameters to explore the design space
2. Model choice - selects the type of surrogate model
3. Model fitting - trains the relevant hyperparameters of the selected model to the data set
4. Model accuracy - verifies that the model is likely by testing the prediction accuracy against the data

Table 4 shows part of the huge variety of sampling methods, model choice and method of fitting decisions that are made for a surrogate. In the experimental design column there are numerous choices but space filling experiments have been found to generate the most accurate surrogate models[146]. Specifically Simpson notes that Hammersley sequences tend to be more evenly distributed in all the dimensions compared to Latin Hypercubes which are only uniform in 1-D projections[146]. However the most common designs include orthogonal arrays, Latin Hypercube designs, Hammersley sequences, and uniforms designs[166]. Kernstine notes that the use of traditional exploration methods may not be well suited for SoS simulations[75]. SoS simulations are usually inherently probabilistic and that it tends to require significant computational resources which limits the ability to run many points.

Table 4: Options for Each Stage of Surrogate Modeling[166]

Experimental Design	Surrogate Model Choice	Model Fitting
- Classic methods	- Polynomial	- (Weighted) Least
* (Fractional)	(linear, quadratic,	squares regression
factorial	or higher)	- Best Linear
* Central	- Splines (linear,	Unbiased Predictor
composite	cubic, NURBS)	(BLUP)
* Box-Belinken	- Multivariate	- Best Linear
*Alphabetical	Adaptive	Predictor
optimal	Regression Splines	- Log-likelihood
* Plackett-Burman	(MARS)	- Multipoint
- Space-filling	- Gaussian Process	approximation
methods	- Kriging	(MPA)
* Simple Grids	- Radial Basis	- Sequential or
* Latin Hypercube	Functions	adaptive
* Orthogonal	(RBF)	metamodeling
Arrays	- Least	- Back propagation
* Hammersley	interpolating	(for ANN)
sequence	polynomials	- Entropy
* Uniform designs	- Artificial Neural	(inf.-theoretic for
* Minimax and	Network	inductive learning
Maximin	- Knowledge Base	on decision tree)
- Hybrid methods	or Decision Tree	
- Random or	- Support Vector	
human selection	Machine	
- Importance	(SVM)	
sampling	- Hybrid models	
- Directional		
simulation		
- Sequential or		
adaptive methods		

The choice of surrogate model is the most critical step. It is possible that the type of model selected is incapable of being satisfactorily trained to the data. For example a linear model would not be able to fit a design space with discrete jumps. The model choices from above all are regression techniques that regress to the mean of the data and would be well suited for capturing the joint distribution of a stochastic point cloud. Mosteller and Turkey note that this is only imparts a piece of the understanding of the problem and that understanding the distribution of the output yields a more complete picture[105]. With a probabilistic regression the goal is to find the probability of y conditioned on x.

$$P(y|x)$$

Probability of y given a value of x is analogous to $f(x)$. It can still be beneficial to have a point value of y and in that case the estimate is to take the expectation of y given x.

$$\begin{aligned} \hat{y} &= \mathbb{E}(y|x=\hat{x}) \\ &= \int P(y|x=\hat{x})\, y\, dy \end{aligned}$$

There are a few choices for estimating this conditional probability. Quantile Regression from Koenker and Bassett attempts to regress specific quantiles from the data that can be used to given an idea of the conditional distribution [85]. They argue regressing the quantiles yields a substantial improvement for linear models with non-Gaussian noise.

Takeuchi modified Quantial Regression with a nonparametric kernel formulation[157]. Kernel quantile regression (KQR) can be used to generate the conditional distribution by solving for all the quantiles. However, the computation can be unstable for some problems.

Bishop proposed the mixture density network (MDN) as a class of neural network extended to represent arbitrary conditional probabilities [16]. The neural network outputs an intermediate parameter vector that is used in a mixture model to output the conditional probability. MDN works well, although its training is time-consuming and limited to only local optimal solutions.

Gaussian processes typically assumes the data as being normally distributed but Tresp introduces a mixture of Gaussian processes to estimate the conditional density[160]. They find it possible to train faster and that it can be used as building blocks in a graphical model.

Sugiyama et al. applied their earlier density ratio estimation to conditional probability[154]. By estimating the conditional distribution directly, they find the Least-Squares Conditional Density Estimation (LSCDE) performed the best against other techniques. All of these density estimation methods look at predicting only a single dimensional output against some number of inputs.

5.2 Probabilistic Graphical Models

Learning a high dimensional joint probability distribution directly is typically not practical and suffers from the curse of dimensionality. Probabilistic graphical models solve this issue by finding an appropriate factorization of the joint probability distribution such that only relevant variable interactions are captured. They offer an intuitive representation of dependencies and compact representation of the joint probability distribution [27]. Again the surrogate model choice needs to be able to handle large dimensions, noisy or probabilistic data that is expected with a generic system-of-systems product family. Graphical models may be a viable alternative to the classical regression approach [165].

Sub-Hypothesis 3.1

If a model can be generated that encodes the joint probability distribution, then component similarities can be inference given performance constraints.

In a probabilistic graphical model, there is a set of nodes (vertices) connected by links (also called edges or arcs)[17]. The graphical representation of probabilistic relationships allows for efficient representation of the dependencies between variables [27]. Conditional independence is captured through the appropriate graph edges between nodes. Due to practicality, the links are limited to only consider strong dependencies between the variables. Furthermore, Probabilistic graphical models confer several advantages over pure algebraic manipulations due to the use of statistical inference techniques.

In creating a probabilistic graphical model there are two steps. First the structure, or how the nodes are connected, must be estimated. Second using the found structure, the conditional probabilities of the various nodes must be learned.

There are two main types of probabilistic graphical models, directed graphs (Bayesian networks) and undirected graphs (Markov random fields). Directed graphs are useful for expressing causal relationships between random variables. Undirected graphs are better suited to expressing soft constraints between random variables[17]. The main distinction between the two is that directed graphs are restricted to exclude all closed paths (called cycles). Markov random fields require more advanced sampling methods when performing statistical inference like Markov Chain Monte Carlo (MCMC), Gibbs sampling, or the Metropolis-Hastings Algorithm. Bayesian networks offer a better starting point for understanding the mechanics of probabilistic graphical models.

5.2.1 Bayesian Networks

The term Bayesian networks was introduced by Pearl[117] as it provides a convenient structure for performing Bayes rule. Bayes rule codifies the update of existing beliefs with new observed evidence, equ 18[10]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{18}$$

There are a variety of subclasses of Bayesian networks: recursive graphical models, Bayesian belief networks, belief networks, causal probabilistic networks, causal networks, and influence diagrams[34]. All Bayesian networks attempt to mimic human's inferential reasoning. Human knowledge is comprised of low-order marginal and conditional probabilities. For instance how likely the grass is wet given that it rained today. A Bayesian network can be considered a probabilistic expert system where the encoded knowledge is contained in the topology of the network and the conditional distributions of each node[58]. Guo notes one of the main uses of the knowledge base is to use it to infer posterior distributions from a particular set of domain queries[58].

Bayesian networks are compact and provide intuitive representation of relationships between the variables[118]. In Bayesian networks the joint distribution has been factored such that the directed links between nodes indicate parent child relationships where the child node has been conditioned on the values from all of it's parents. As an example consider a joint probability distribution of three variables a, b, and c being $P(a,b,c)$. From the product rule for probability this joint distribution may be factored as [17]

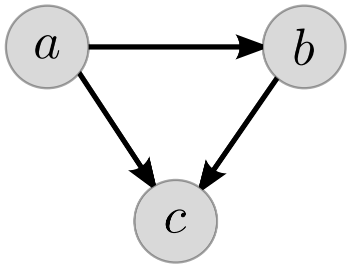


Figure 29: Directed Graph Example

$$\begin{aligned} P(a,b,c) &= P(c|a,b)P(a,b) \\ &= P(c|a,b)P(b|a)P(a) \end{aligned} \tag{19}$$

Graphically, this factoring is shown in figure 29, and represents the joint probability distribution. Each node is the conditional distribution given the values of their parents (all the other nodes that point into this node). An important qualification is that a different factoring of the joint distribution would show a different graph and may be a more accurate representation of the data.

In general for a network with K variables in the domain the joint distribution is Equ 20[17].

$$P(\bar{x}) = \prod_{k=1}^{K} P(x_k|pa_k) \tag{20}$$

where pa_k denotes the parents of x_k and $\bar{x} = \{x_1,...,x_k\}$. With Bayesian networks there is the idea of conditional independence. For instance, two sets of variables x_a and x_b are conditionally independent given a third set of variables x_c if

$$P(x_a,x_b|x_c) = P(x_a|x_c)P(x_b|x_c)\ \forall x_c \tag{21}$$

is equivalent to:

$$P(x_a|x_b,x_c) = P(x_a|x_c)\ \forall x_c \tag{22}$$

Conditional independence allows for simplifying both the structure of the model and the computations needed to perform inference and learning under that model[17]. A node is conditionally

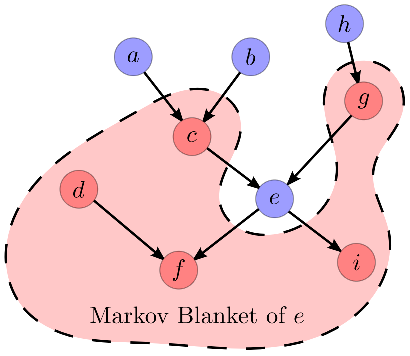


Figure 30: Markov Blanket

independent of all other nodes in the network given its Markov blanket. The Markov blanket includes all of the node's parents, children, and children's parents, figure 30.

Hierarchical modeling is fundamental to Bayesian statistics[158]. Hierarchical Bayesian networks (HBN) are an extension of Bayesian networks whereby the nodes are aggregations of simpler nodes. In fact, every node could itself be another HBN[59]. A typical example of hierarchical Bayesian model is when the node's distribution includes hyperparameters like a normal distribution's mean and standard deviation. These hyperparameters then may be described by other nodes with may have additional hyperparameters to describe their shape. The benefit of HBN is the higher order nodes can be decomposed by their hyperparameters until it reaches the atomic level where the hyperparameter can be represented with a simple marginal distribution[158]. The ability to decompose the network down to simpler levels may eventually yield hyperparameters that have operational meaning and help in fitting the model.

Another class of Bayesian networks is Dynamic Bayesian networks (DBN). DBN capture temporal processes by constraining the structure to specify how variables change in time[34]. Networks of this type have been applied to control problems.

5.2.1.1 Graphical Model Structure Learning

As mentioned previously the structure of the network is critical. Having only the critical links allows probabilistic graphical models to represent large joint distributions more compactly. Incorporating insignificant connections leads to over fitting of the model while failing to have the important connections leads to a network that does not accurately model reality.

Table 5: Number of Possible DAGs for a Given Number of Variables

Number of variables in DAG	Number of possible DAGs
1	1
2	3
3	25
4	543
5	29,281
6	3,781,503
7	1,138,779,265
8	783,702,329,343
9	1,213,442,454,842,881
10	4,175,098,976,430,598,100

The problem of learning the network structure is shown to be NP hard[25]. This is due to another huge combinatorial space. Robinson gives the number of possible DAGs combinations for the structure space of n nodes[129]:

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} C_i^n 2^{i(n-i)} f(n-i) \tag{23}$$

Table 5 show how quickly the possible DAG combinations grows. Because of the large combinatorial structure space, brute force methods are computationally prohibitive requiring heuristic methods to find a reasonable network candidates[17].

An exhaustive search brute force approach would find the global optimal structure. However it requires the enumeration and testing all possible graphs and is infeasible on all but the smallest networks. Fortunately there are a variety of methods to learn the structure of the network. Daly divides these methods into three main if slightly overlapping methods: dynamic programming, heuristic score and search, constraint-based[34]. In dynamic programming, a score is calculated for all small numbers of variables and combine these models.

Heuristic score and search methods are the most widely used methods. All of the heuristic algorithms are comprised of[34]: a space of allowable states each representing a possible network structure, a mechanism to encode these states, a mechanism to transition from state to state in the space, and a scoring function to compare states against each other. The most basic heuristic algorithm is the greedy search. The greedy search is analogous to a hill climbing optimizer. Links are added and removed until the score is improved over the current value. This process continuous until no link changes yield a score improvement. Another famous heuristic method is the K2 algorithm introduced by Cooper and Herskovits[29]. The K2 algorithm uses a hill climbing approach but is restricted by the order of the nodes. The node order denotes preceding nodes as possible parents to later nodes. This restriction can be useful but in the product family design a good node order may not be known. Other heuristic optimization methods like genetic algorithms, simulated annealing, particle swarm optimization can also successfully identify Bayesian network structure. Chickering shows combining random restarts with the greedy search produces better results than simulated annealing for the same computational time[26].

Constraint-based methods determined the Bayesian network structure through the use of dependency tests. These tests then can serve to limit the number of possible links to evaluate. There are many statistical tests that could be used to calculate the dependency, χ^2, maximum likelihood statistical significance or G-test, mutual information, and conditional mutual information[124]. Conditional independence tests allows edges to be removed from the network making it easier to train. No matter the condition independence test, constraint-based methods have some overlap with the heuristic score and search. A scoring function is still needed to compare alternative structures.

In order to determine what is a "good" network, a scoring metric is used. This measures the quality of the network and is typically proportional to the likelihood function and a penalty that grows for the number of links (limit complexity)[120]. The metric could also include results from statistical tests on independence. Although the structure of the network does not directly depend on the choice of node's conditional distribution, that choice could affect the selected scoring function.

The simplest scoring metric is the maximum likelihood score. Because the likelihood involves multiplying many small numbers it can encounter numerical problems. This is helped by working on the log of the likelihood so given data D and model B_s with hyperparameters θ the log likelihood assuming each data sample generated independently is

$$\log P(D|\theta, B_s) = \log \prod_{j=1}^{k} \prod_{i=1}^{n} P\left(x_{ij}|Pa_{x_{ij}}, \theta, B_s\right) \tag{24}$$

where n is the number of variables in the network and k is the number of samples in D and $Pa_{x_{ij}}$ is the parents of x_{ij}. This score though has a tendency to include more links than necessary resulting in over fitting. To modify this there is the Bayesian information criterion (BIC) among others. These metrics typically still calculate the likelihood but add some penalty for increasing model complexity.

$$BIC\left(B_s, D\right) = \log P\left(D|\theta, B_s\right) - \frac{d}{2}\log\left(n\right) \tag{25}$$

where θ is the maximum likelihood of the parameters in the network B_s, d is the number of parameters in the node, n is the sample size. The Minimum description length (MDL) is a modified BIC with yet another penalty for complexity[30].

$$MDL\left(B_s, D\right) = \log P\left(D|\theta, B_s\right) - \frac{d}{2}\log\left(n\right) + C_k \tag{26}$$

where $C_k = \sum_{i=1}^{k}\left(1 + |Pa_{x_i}|\right)\log k$, and $|Pa_{x_i}|$ is the number of parents for node x_i and k is the total number of variables. To complicate structure learning, not all of the variables may be known for some general Bayesian networks. These latent variables or hidden variables can be crucial to some models[6]. A popular method of determining the conditional probabilities of the latent variables is the EM-algorithm [38]. Daly points out that it always converges to at least a local optimum[34].

5.2.1.2 Conditional Density Estimation

Once a network structure has been established the second step for fitting a Bayesian network is to find the condition densities for every node. In general, a node could be either continuous or discrete. If a Bayesian network has both types of nodes then it is a mixed or hybrid Bayesian network.

Discrete networks are based on multinomial distributions and can be generated by calculating a conditional probability table (CPT). The CPT is calculated by counting the number of instances that variable's outcome happens given it's parents. Continuous Bayesian networks are more difficult than discrete networks and it is not uncommon to discretize a continuous domain. One difficulty with discretization is the loss of information, and that if the resolution is too fine for the number of samples, there will not be any outcomes in the bin to count. This can be a significant problem for higher dimensional nodes. This state space explosion does not happen if the domain remains continuous.

Instead of a simple counting problem, the difficult problem of conditional density estimation is created. Now that the conditionals are more difficult to calculate than CPT, some additional decisions must be made. If the kind of distribution is known (e.g. normal) then all that is required is to fit the parameters of the distributions. However, these parametric techniques can be restrictive if the data does not follow a traditional distribution (as in the case for SoS problems). Because the distribution is not known a priori, nonparametric techniques maybe applied which make less assumptions about the density of the data[50]. Ickstadt discusses nonparametric Bayesian networks using mixtures of distributions[70]. Formally, the conditional distribution of y given x is

$$P(y|x) = \frac{P(x,y)}{P(x)} \tag{27}$$

Given the way the conditional density is calculated, quotient-shaping is an approach that estimates $P(x,y)$ and $P(x)$ independently and then divide the result[50]. This simple approach tends to magnify errors in the resulting conditional density.

Another nonparametric conditional density estimation approach is the quantile-copula proposed by Faugeras[42]. A copula is a description of the cumulative multivariate distribution function. To accomplish this the variables are represented by their marginal distributions and the copula describes their interaction. Elidan extensively uses copulas in learning Bayesian networks[40]. The benefit by using copulas is the probability distributions can be efficiently sampled using transformation sampling. Because the CDF varies uniformly between 0 and 1 in transformation sampling can generate a value between 0 and 1 then use the CDF to find the corresponding value for the variable.

As mentioned previously it has been found that LSCDE outperformed other conditional density estimation techniques, although it has not been compared with copulas. In this dissertation, LSCDE will be used as a nonparametric technique to estimate the condition distributions for the nodes in a continuous Bayesian network. The following formulation is from Sugiuama and recreated here for the purposes of completeness[154]. In LSCDE, instead of estimating the numerator and denominator independently, it is considered proportional to the density ratio

$$P(y|x) = \frac{P(x,y)}{P(x)} \propto r(x,y) \tag{28}$$

The density ratio, $r(x,y)$, is a linear model of

$$r(x,y) = \alpha^T \bar{\phi}(\hat{x},y) \tag{29}$$

where $\phi(x,y)$ are basis functions and α are their associated weights. These weights, $\tilde{\alpha}$, are calculated as the solution to minimize the squared error.

$$\tilde{\alpha} = \left(\hat{H} + \lambda I_b\right)^{-1} \hat{h} \tag{30}$$

where

$$\begin{aligned}
\hat{H} &= \frac{1}{n}\sum_{i=1}^{n} \Phi(x_i) && (31)\\
\hat{h} &= \frac{1}{n}\sum_{i=1}^{n} \bar{\phi}(x_i,y_i) && (32)\\
\Phi(x) &= \int \bar{\phi}(x,y)\,\bar{\phi}(x,y)^T\,dy && (33)
\end{aligned}$$

Because the density ratio must be positive any negative weights $\tilde{\alpha}$ are pruned.

$$\hat{\alpha} = \max(0,\tilde{\alpha}) \tag{34}$$

Finally, the conditional probability is equal to the normalized density ratio

$$P(y|x=\hat{x}) = \frac{\hat{\alpha}^T \bar{\phi}(\hat{x},y)}{\int \hat{\alpha}^T \bar{\phi}(\hat{x},y)\,dy} \tag{35}$$

The kernel used for the basis functions $\phi(x,y)$ will be discussed in the next chapter. Once both the network structure is learned and the node conditional distributions are estimated the training is complete. In other words, the completed Bayesian network has been fit to the joint probability distribution of the training data.

5.2.2 Inference

The purpose of building this probabilistic surrogate is to find the design variables probability distributions given other nodes are set to certain values. This is similar to traditional functional surrogate models, $y = f(x)$, where the goal is to calculate y for various values of x. Inference is the process

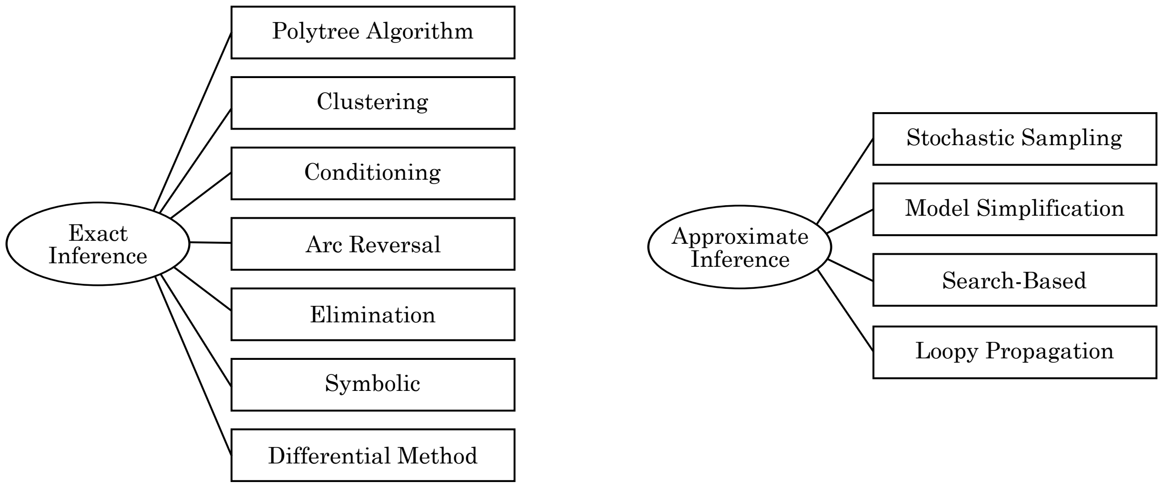


Figure 31: Inference Techniques[58]

to determine node posterior distributions conditioned on a set of evidence nodes[108]. However inference in a Bayesian network is much more flexible because the evidence nodes can be any subset of the network's nodes. Consider the following conditional probability query[87]

$$P(\mathbf{X}|\mathbf{E}=e) \tag{36}$$

where **E** are the evidence nodes with observed values *e*, and **X** are the network nodes being queried. In effect, the evidence nodes are "clamped" to their respective values *e* while the other network nodes will reflex the likely values that correspond to the evidence. There are numerous different techniques for performing the inference but can be divided into two main branches exact and approximate, figure 31.

Exact inference methods yield an exact distribution for the variables. These methods usually attempt to simplify the network by removing unimportant nodes, collapsing several nodes, and reversing the parent child relationships. In arc reversal, the direction of the edges are adjusted so that the evidence nodes are treated as parents. Node reduction is a method to remove nodes that are highly dependent on neighboring nodes thereby simplifying the network. For example in figure 32, node *b* is highly correlated with node *c* and could be summed out.

It can be difficult to compute the posterior distributions exactly either because of high dimensionality, the continuous variables, or required integrations may not have closed form analytical solutions. In general the problem of inference in Bayesian networks is NP hard which makes exact

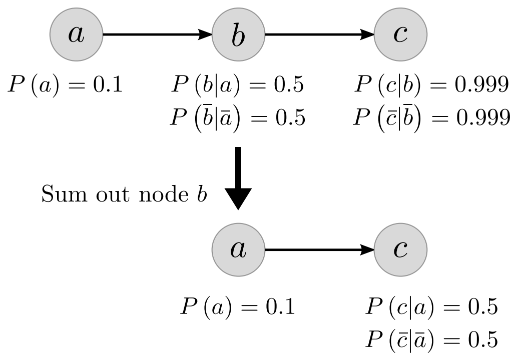


Figure 32: Example of node Reduction[27]

inference inefficient or impossible for large scale networks[108]. Thus if exact inference cannot be performed, then approximate inference must be used. Most commonly used approximate inference technique are stochastic Monte Carlo sampling methods.

In these methods, samples are simulated from the network according to the probabilities of the conditional probability query. The frequencies of these samples then serve as an approximation for the posterior distributions. These sampling methods can be relatively slow to converge, but exact given an infinite amount of computation time. It is imperative to speed up the convergence rate of this class of methods in order for them to find more practical applications[21]. Interestingly, the convergence rate for these sampling methods asymptotically is $\frac{1}{n^{1/2}}$where n is the number samples[94].

The most basic form of stochastic approximate inference sampling is the logic sampling also known as ancestor sampling or forward sampling[27]. Logic sampling, similar to rejection sampling, generates a samples from the network node i given its parents $P(x_i|pa_i)$, even if that node is in the evidence set and has $P(E_i)$. To correct for this, some samples get rejected according to $u < \frac{P(E_i|pa_i)}{MP(s_i|pa_i)}$ where the rejection rate u is uniformly distributed between 0 and 1.

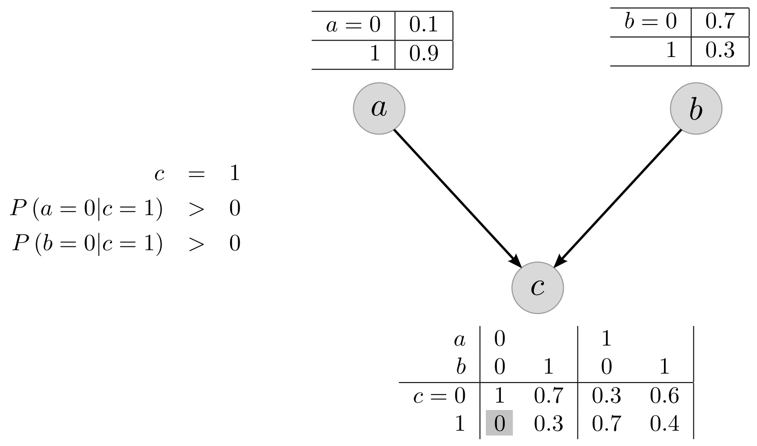
In performing logic sampling the nodes in the Bayesian network are sorted topologically. First the lowest numbered node, the one with no parents, is sampled. The network nodes are then progressively sampled so that all of the parents of a node are known first so that nodes conditional density can be evaluate. Because evidence nodes are not locked to the observed values, samples are continually generated until they agree with the evidence nodes. Samples that do not agree will be discarded. Algorithmically logic sampling for the Bayesian network B_s is:

Algorithm 2 Logic Sampling

```
Topologically sort nodes in network $B_s$
for each node $i \in B_s$ do
    $s_i \leftarrow P(x_i|pa_i)$ {Generate a test x for node $i$ from the conditional distribution}
    if $i \in E$ node is in the set of evidence then
        $u \leftarrow \mathscr{U}[0,1]$ {Rejection rate $s_i$}
        if $u < \frac{P(E_i|pa_i)}{MP(s_i|pa_i)}$ then
            $s_i \in S$ {Add node sampled to the sample set}
        end if
    else
        $s_i \in S$
    end if
end for
```

Logic sampling is only efficient when the evidence is very likely and the evidence nodes have few parents. One method, backwards sampling, helps improve the efficiency by reversing network arcs so that connections flow away from any evidence nodes[53]. However, depending on the structure and the number of evidence nodes it may not be possible to eliminate all of the evidence nodes' parents. Because samples are discarded, an inordinate number of simulations may be needed to generate one instance conforming to the evidence[63].

To overcome this difficulty it is beneficial to sample directly from the evidence nodes' observed values. Effectively the samples are going to be generated in more likely or important regions[97]. However sampling from the evidence nodes directly requires the introduction of sample weights[108]. These weights, w_i, correct for sampling from the evidence instead of the network. Shachter and Fung first used likelihood weightings from Simulation approaches for network inference[137, 54].



$$c = 1$$
$$P(a=0|c=1) > 0$$
$$P(b=0|c=1) > 0$$

$a=0$	0.1
1	0.9

$b=0$	0.7
1	0.3

a	0		1	
b	0	1	0	1
$c=0$	1	0.7	0.3	0.6
1	0	0.3	0.7	0.4

Figure 33: Example Importance Sample with CPT[27]

Algorithm 3 Importance Sampling

```
Topologically sort nodes in network
for each node $i \in B_s$ do
    if $i \in E$ node is in the set of evidence then
        $s_i \leftarrow P(E_i)$ {Generate a test x for node $i$ from the conditional distribution}
    else
        $s_i \leftarrow P(x_i|pa_i)$ {Generate a test x for node $i$ from the conditional distribution}
    end if
    $s_i \in S$
    $\hat{w}_i = \frac{P(x_i|pa_i)}{P(E_i)}$ {Cooresponding score of sample i}
end for
$w_i = \frac{\hat{w}_i}{\sum_i \hat{w}_i}$ {Normalize sample weights }
```

Likelihood weighting can still reject cases if the weighting is zero. An example is shown in figure 33. In the sampling procedure, a would be sampled and then b would be sampled. However, if both are zero and since $c = 1$ the sample weight would also be 0. This is because there is no chance the network could have generated it and it would be rejected, $P(c = 0|a = 0, b = 0) = 0$.

Because all the samples are used the predicted distribution converges more quickly compared to logic sampling. However it is still possible for unlikely evidence to have most of the sample weights be zero except and the occasional large weighted sample dominating the other samples[169]. This is especially an issue for high dimensional problems, complex conditional distributions, extremely unlikely evidence, and when low numbers of samples are taken[126].

Importance sampling works because the samples are drawn from "information packed" sample space[21]. There are several studies that show improvements compared to basic importance sampling by further refining the ability to target this informative space. Cheng and Druzdzel propose an Adaptive Importance Sampling for Bayesian networks (AIS-BN)[24]. Their key to speeding up the convergence is to iteratively sample while updating the sampling distributions and eventually the sampling distribution converges to the correct posterior. The AIS-BN introduces different sample weights depending on the different learning stages and the sampling distribution is updated[58].

Another improvement to importance sampling is self importance sampling SIS[137]. The other approximate inference algorithms like Loopy Belief Propagation can be expressed in terms of the propagation of local messages around the graph [17]. These kinds of message passing inference algorithms begin by creating a factor graph. A factor graph is a undirected graph where nodes either represent the variables or the edges in the original DAG. These new edge node types serve as factors and help control the algorithms message passing. The benefit of message passing is unlikely evidence can back propagate more easily but can also cause the network to oscillate between states as conflicting evidence is passed around. Yuan proposes an evidence prepropagation importance sampling EPIS-BN[169]. In this they use a two stages, first a loopy belief propagation to initialize the network then an epsilon cutoff heuristic to prune unlikely probabilities. Overall this improves the sampling importance function and increases accuracy in the posterior distributions.

5.3 Similarity Measures

In the context of family design these posterior distributions for the design variables represent the likely design variable settings for the different components. To compare the different products components then there needs to be some similarity metric between different module posterior distributions. There has been much work into the area to measure the similarity between two probability distributions.

To reiterate from the similarity measure in the previous chapter there are several properties that a metric needs to possess. It must be symmetric, and obeys the triangle inequality. The Kullback-Leibler and Jeffrey divergences are information theory based. The KL divergence measures how well one distribution represents another. However the KL divergence is not symmetric. Jeffery divergence is a modification from the Kullback-Leibler divergence

$$d(P,Q) = \int (P(x) - Q(x))(\ln P(x) - \ln Q(x))\, dx \tag{37}$$

The Hellinger distance is another metric for evaluating how far one probability distribution is to another[49].

$$h(P,Q) = \frac{1}{\sqrt{2}} \cdot \int \sqrt{P} - \sqrt{Q} \tag{38}$$

$$h(P,Q) = \frac{1}{\sqrt{2}} \cdot \int \left(\sqrt{f(x)} - \sqrt{g(x)}\right)^2 dx \tag{39}$$

Benefit is that distance $0 \le h(P,Q) \le 1$. zero for identical and 1 for no commonality. This metric also satisfies the triangle inequality. The Hellinger distance is closely related to of the Bhattacharya coefficient which lacks satisfying the triangular equality.

Rubner introduces the earth movers distance (EMD) as a metric for comparing multidimensional distributions[131].

$$\boxed{\begin{aligned}
&\text{minimize } WORK(P,Q,F) = \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} d_{ij} \\
&\text{with respect to } F = \{f_{ij}\} \\
&\text{subject to } f_{ij} \ge 0 && 1 \le i \le m, 1 \le j \le n \\
&\qquad \sum_{j=1}^{n} f_{ij} \le w_{p_i} && 1 \le i \le m \\
&\qquad \sum_{i=1}^{m} f_{ij} \le w_{q_j} && 1 \le j \le n \\
&\qquad \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = \min\left(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_j}\right)
\end{aligned}}$$

where:

$D = [d_{ij}]$ ground distance matrix

$F = [f_{ij}]$ flow between the two probability distributions

Effectively, this describes the minimum amount of work needed to transfer one pile of dirt into another pile of dirt. The final distance then is normalized:

$$EMD(P,Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} \tag{40}$$

Because there is a linear programming step the computational cost for the EMD is high. The EMD is a flexible metric that outperforms other dissimilarity measures[131].

Levina and Bickel find that the EMD is equivalent to the mallows distance but is more robust implementation[93]. EMD has been used with different methods of calculating the ground distance matrix[8].

5.4 Bayesian Network Applications

Now that Bayesian networks have been introduced along with the theory of building and using them, this section serves to introduce current uses. Computational resources continue to improve allowing more Bayesian networks to be used in more complicated problems.

Bayesian networks have been used in a type of optimization algorithm called Evolutionary Distribution Algorithms (EDA). EDA construct a probability distribution over the search space of an optimization problem and adaptively learn while driving the search process[79]. One particular type of EDA is the Probabilistic Model Building Genetic Algorithm or PMBGA. In a PMBGA the process of mutation and crossover is replaced with generating a probabilistic model that can be sampled from to generate the next generation.

The end result of the PMBGA is a probabilistic model that can be sampled to generate all global optima[120]. Frequently a subject matter expert has significant knowledge of their problem domain. It would be wasteful for the optimization not to include and exploit this knowledge. Fortunately, probabilistic models provide a flexible framework for any incorporation of prior knowledge. One way this can be accomplished is with biasing the initial sample population[120]. Additionally having prior knowledge about the interactions between variables can lead to improved structure quality and better improve the efficiency of the optimization. Furthermore, the structure of the probabilistic model does not have to be retrained with every new generation and can build on previously learned structures.

Another interesting application is presented by Gallagher[55]. Here Bayesian inference is performed between the iterations to help guide the evolution by combining the probabilistic model with prior distributions and improve the optimization convergence. Ahn introduces a MultiObjective Real coded Bayesian Optimization algorithm (MrBOA)[5]. In this study the NSGA-II method of determining dominance is used to help progress the optimization to the Pareto frontier.

Outside of optimization, Bayesian networks are used extensively as knowledge bases. Most commonly as medical diagnoses. One recent study in the field of engineering by Kestner et al. uses exact inference on discrete Bayesian network for complex conceptual systems[80]. In the study, they performed a trade study and subsystem design of directed energy system.

5.5 Bayesian Network Approach

This section will now establish the Bayesian network process that will be used in the product family design methodology. The end goal is to identify component commonality between a set of products. This approach was formulated due to some limitations noted with the previous data mining approach. These issues include the requirement of a large exploration of the design space. This exploration needs to not add any artificial groupings of points so that the clustering approach finds relationships between the different product variants that exist solely because of the physics of the design problem. The need to not bias the database limits the kinds of explorations; adaptive sampling and other evolutionary optimization strategies build new data from those previously found better performing designs. Unbiased explorations would be Design of Experiments and random Monte Carlo approaches. The downside of these exploration methods is that a significant amount of effort is spent computing regions of the design space that is either infeasible or significantly dominated instead of regions of interesting trade-offs.

To remedy these limitations, a probabilistic approach is formulated. This approach attempts to build a probabilistic Bayesian Network model of the Joint Probability Distribution of a design space exploration. The method used for finding the network structure is to use the BIC with a greedy search. Like the previous fuzzy clustering approach, the design space needs to be explored first. The options available for this can the same as the clustering approach.

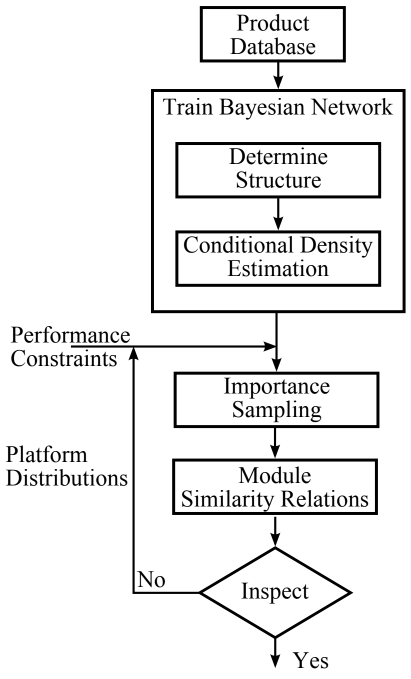


Figure 34: Bayesian Network Process

After the space has been explored the Bayesian network needs to be created. To learn the network structure this approach proposes to use a greedy search algorithm that uses the BIC score to help limit excess links between the nodes. The BIC score requires the likelihood of the data for the given structure and conditional density so the nodes need to be fitted as well. This does add to the training time however and there could be some benefit to including some statistical independence tests to help guide the search. The greedy search was chosen because it will at least yield a local optimum and can be restarted with different initial conditions to help select better final structure. If there is a subject matter expert available, they can provide feedback as to which structures have justifiable links between the variables and can prune away or add as necessary.

Once the structure is learned, this approach proposes to estimate the conditional distributions using the LSCDE process outlined above. It is a reasonably robust and quick method that yields better final fits comparing to several other alternative methods. There is a hyperparameter search that will be discussed in further detail in the following chapter. The main purpose of building the Bayesian network is to perform inference. Performance constraints can be applied as evidence and the inference propagates these through the network and yields the posterior distributions for the nodes.

The posterior represents the distribution of that variable given the set of evidence. From the family perspective if two products' components have a similar posterior, then they would generate a better platform than if there were dissimilar. The earth movers distance is calculated between each product for each module space. These new module similarity relations now can be visualized and inspected by a decision maker on possible family platform.

The outputs of the method are inter-product similarity for the components. In addition the results of the Bayesian inference enable some additional visualizations that help the designers make informed commonality decisions. Finally with these decisions the impact of the platform configuration can be assessed and more easily traded against other alternatives with performance estimates obtained through subsequent inferences. Additionally to assess the flexibility of a proposed platform the inference can be adjusted as described in sub-hypothesis 3.2.

Sub-Hypothesis 3.2

If the design constraints are changed, then any changes to the posterior distributions from the probabilistic model will reveal the sensitivity of component sharing

The Bayesian approach appears to be superior to the fuzzy clustering approach as was formulated because of the identified possible limitations from the initial pattern recognition fuzzy clustering approach. Because the Bayesian Network provides a robust framework for inference, it allows the model to be utilized like traditional surrogate models to move along the frontier. Unlike the refiltering and reclustering necessary in the previous data mining approach, the Bayesian inference already has the module variables coupled and is a more direct framework for product family reasoning.

As components are combined, decision makers must accept a trade-off of individual product's performances. The ideal amount of this trade-off is going to be problem specific and will be determined by market forces, i.e. how much money can be saved by making a component common and where exactly the other products must be changed.

In the aerospace industry, the products are large and expensive with the life cycle costs dominated by operational activities. Lowering acquisition costs through better component commonality is important but not while sacrificing individual product performances to a degree that increases operational costs. For instance, an airliner is not going to accept a huge performance penalty in fuel

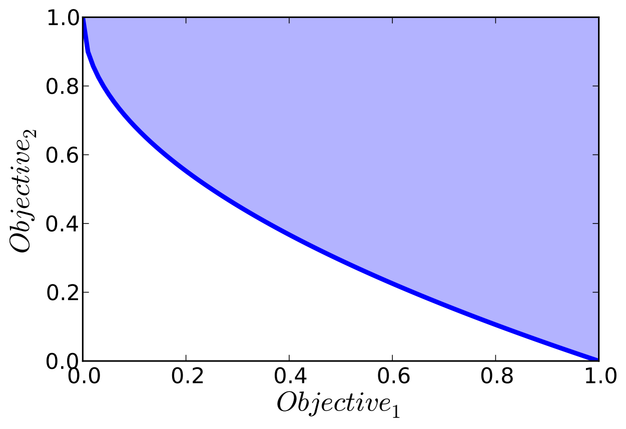


Figure 35: Pareto Frontier

economy if there is only a marginal acquisition cost benefit or maintenance benefit. For these types of products the best families are ones that exploit commonality while minimizing the individual products deviation from their respective Pareto frontiers. Furthermore, the kinds of problems at the conceptual aircraft design level are inherently probabilistic due to limited knowledge of operations for example fuel cost or variations to the mission requirements. This means that even with shared components, the aircraft remain near their respective Pareto front.

The scalability issue that limits the clustering approach stems from the ability to get a large number of points that meet several different filtering criteria. Furthermore, the use of static DoEs to explore the design space leads to significant computational effort spent on regions that are either dominated or infeasible points, figure 35. Because many points fail to lie near the Pareto frontier, they will not contribute to the commonality identification effort and offer little usable information. The data needs to be concentrated in the region of interesting trade-offs. Even with focusing of the Pareto frontier, the clustering approach still requires the space to be dense and care needs to be taken so that groups found through the clustering are natural rather than artifacts of the method to generate the Pareto frontier.

However, if a model can be generated that accurately captures the space, then that model can be evaluated instead of using the precalculated design cases. This interpolation may then be processed to identify patterns in an analogous method to the previous approach and serves as a basis for the probabilistic approach. While there still needs to be sufficient exploration of the design space, the probabilistic model should prove to be more accurate.

In a family that needs to exploit commonality without sacrificing significant individual performance, like in aerospace, the individual performances will remain near their Pareto frontier.

Also the ability to build a probabilistic surrogate capable of capturing the joint probability distribution of a design space has far reaching affects. One other limitation of traditional methods is they are a mapping from a set of inputs to a set of outputs. In a design problem, frequently there is a particular performance desired while the combination of input design variables is not known. Inverse design problem has been solved using surrogates by generating many points and filtering away undesirable points to understand the regions of the feasible design space[14]. The next Chapter 6 discusses in more detail the applicability of generating and using a Bayesian Network surrogate as well as some implementation details.

Chapter VI

FEASIBILITY OF BAYESIAN NETWORK SURROGATE

The previous chapter introduced Bayesian network probabilistic graphical models and formulated how they can be used to perform inference and detect commonality for a family of products. This chapter expands on the theory and discusses the implementation details for learning the Bayesian network structure, fitting the conditional density estimates, and performing inferences. Two test problems are presented that test the ability of the Bayesian network to act as a surrogate model. The first test is an optimization problem that demonstrates the Bayesian network's ability to find and encode the joint distribution of a Pareto frontier. The second test problem uses a Bayesian network to capture the joint distribution of the Breguet range equation applied to design space exploration. This problem also serves to demonstrate the capabilities of importance sampling inference.

6.1 Conditional Density Estimation

As mentioned in the previous chapter, each node in the Bayesian network are conditioned on its parents. Although for the product family design problem the platform configuration is discrete, the design parameters that describe the components is continuous. Frequently in continuous domain Bayesian networks, the nodes are discretized so the traditional conditional probability table can be used. This table is calculated by binning the state space of that node to some allowable resolution and counting the frequency of points that fall into each bin. However, as the node's dimensionality and resolution increase the size of the state space can become large and the number of points that fall into a particular bin become fewer. Too fine of a resolution can limit the accuracy of this method.

Another common method of working with continuous variables in Bayesian networks is to assume the data is from a known parametric distribution such as Gaussian, Poisson, or Dirichlet. This has several drawbacks if the type of distribution is not known or is not from a traditional distribution; like in the case of complicated system-of-systems models. Fortunately, there are a myriad of non-parametric models that have the ability to accurately represent these complicated distributions[70]. The one selected from the last chapter is the Least Squares Conditional Density Estimation (LSCDE)

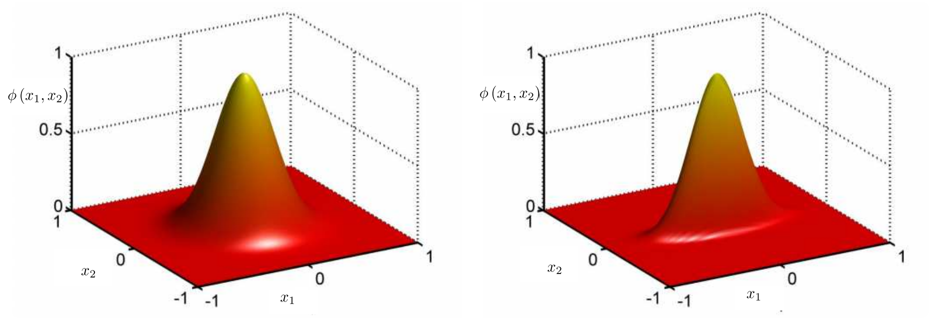


Figure 36: Isotropic Gaussian (left), Anisotropic Gaussian (right)

method from Sugiuama et al[154]. LSCDE estimates directly the density ratio $p(x,y)/p(x)$ which is the conditional distribution of y given x, equ 27. The density ratio is modeled using a mixture of kernel basis functions, equ 29. A kernel function is a real-valued function of two arguments $\phi(x,x')$ and is typically symmetric and positive[106]. Perhaps the most recognizable kernel function is the Gaussian kernel[41]. The general anisotropic form is

$$\phi\left(\bar{x},\bar{x}'\right) = \exp\left(-\frac{1}{2}\left(\bar{x}-\bar{x}'\right)^T \Sigma^{-1}\left(\bar{x}-\bar{x}'\right)\right) \tag{41}$$

where Σ is the covariance. If the covariance is diagonal, the kernel becomes isotropic and simplifies to

$$\phi\left(\bar{x},\bar{x}'\right) = \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{\left(x_i-x_i'\right)^2}{\sigma_i^2}\right) \tag{42}$$

Figure 36 shows a comparison between the shapes of the isotropic and anisotropic Gaussian kernel. The anisotropic kernel is more focused along the direction of the covariance.

For Sugiuama's implementation of the LSCDE, the isotropic Gaussian kernel is selected, equ 43[154]. The major benefit of using an isotropic Gaussian is that the integrals in equ 33 and equ 35 can be analytically evaluated and leads to faster computation times.

$$\phi_l(x,y) \quad = \quad \exp\left(-\frac{\|x-u_l\|^2}{2\sigma^2}\right)\exp\left(-\frac{\|y-v_l\|^2}{2\sigma^2}\right) \tag{43}$$

To understand the implications of this kernel choice, the following optimization problem will be used. ZDT1 is a multiobjective problem function that allows the testing of the kernel used for the density ratio estimation in the Bayesian network approach[22]. One of the benefits of this test function is it can be scaled to some number of input $\bar{x}$ dimensions n.

$$\begin{aligned}
\text{maximize } & ZDT1.f_1(\bar{x}) = x_1 \\
& ZDT1.f_2(\bar{x}) = g(\bar{x})\left(1 - \sqrt{\frac{x_1}{g(\bar{x})}}\right) \\
\text{with respect to } & \bar{x} = \{x_1, x_2, ..., x_i\} \\
\text{subject to } & g(\bar{x}) = 1 + 9\sum_{i=2}^{n} \frac{x_i}{n-1} \\
& 0 < x_i < 1
\end{aligned}$$

It has been observed that even with the product family commonality trade-off, the products are still approximately Pareto optimal. This means that the joint probability density function the Bayesian network may need to encode is the Pareto frontier. To represent the Pareto frontier, here are a few behavioral properties that are necessary for the Bayesian network to capture. Specifically, along the Pareto frontier, there can be strong correlations between the different variables. This behavior will guide the selection of the kernel basis function used in the LSCDE.

If the Bayesian network can be used in a PMBGA to find the Pareto frontier and it can be sampled from to recreate the Pareto frontier, it supports the feasibility of the Bayesian network approach for product family commonality. The goal of this test optimization is not to prove that the Bayesian PMBGA is the fastest executing optimization, or that it converges in fewer generations than other methods. Instead, the goal is to support the claim that selecting the correct kernel function allows the Bayesian PMBGA to recreate a Pareto frontier.

Before using the PMBGA, the kernel functions need to be tested. To do that, the ZDT1 function is optimized using a standard NSGA-II. The result of this optimization is shown in figure 37. The scatterplot matrix shows the Pareto frontier between f_1 and f_2. The main features of this data set are not only the strong relationships between x_1 and the objective functions f_1 and f_2 but also the values of x_2, x_3 and x_4 all being small. The Bayesian network with isotropic Gaussian kernels is fit using the found Pareto frontier data.

The first step is to identity the key relationships between the different nodes in the network. This

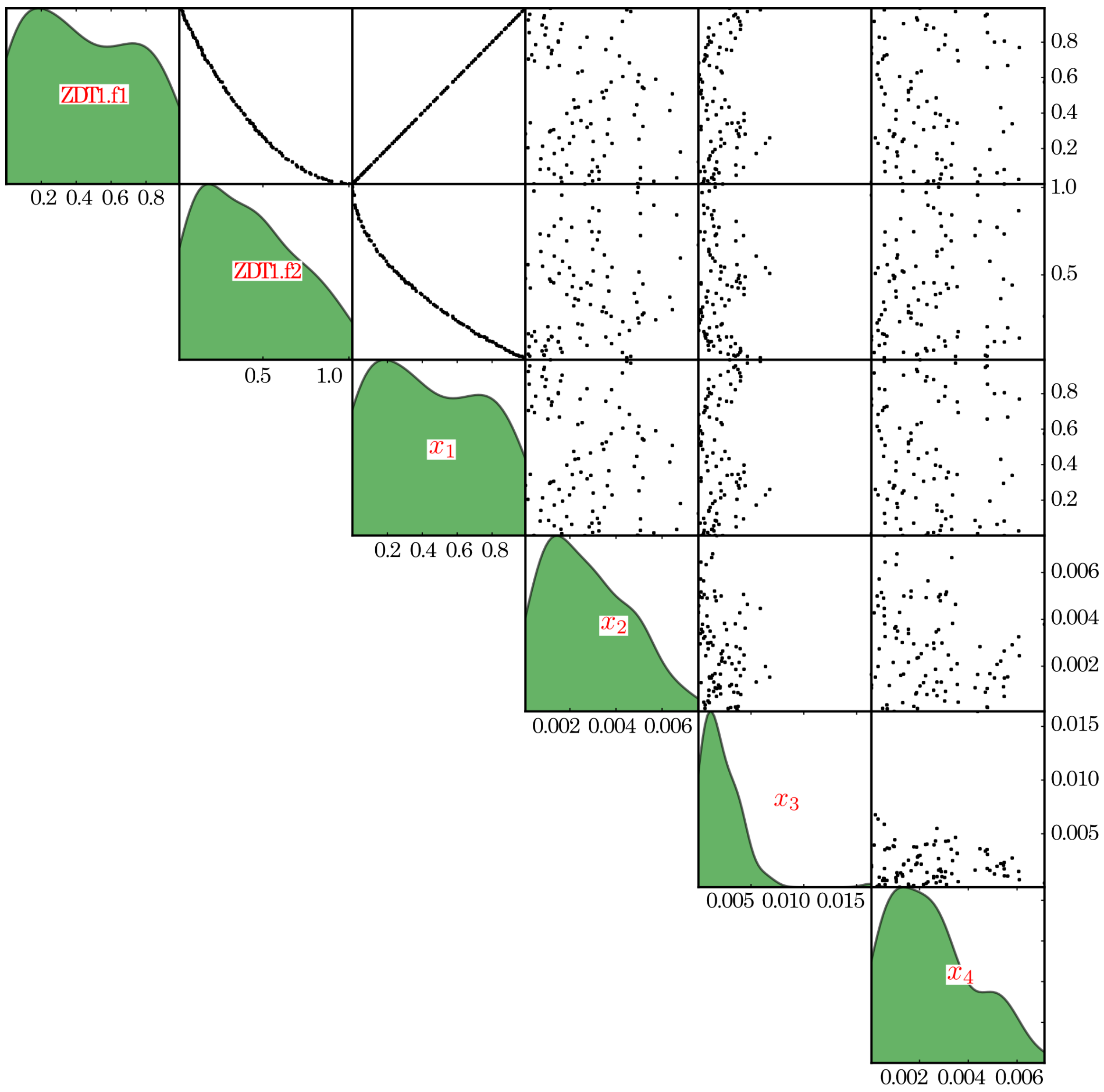


Figure 37: Pareto Front of Test Function

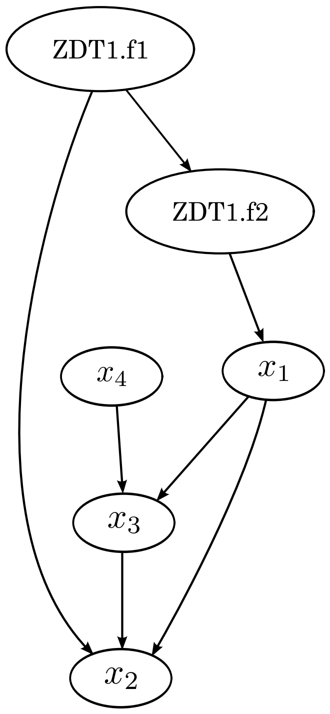


Figure 38: Network Structure for Pareto Frontier of Test Function

is again done using a greedy search with several random restarts and calculating the BIC score to limit overfitting. Figure 38 shows the resulting learned network structure. As expected, the main relationship is accurately accounted for with the links from f_1 to f_2 and f_2 to x_1.

Now that the network structure has been determined, the following conditional densities need to be calculated from the data: $P(f_1)$, $P(f_2|f_1)$, $P(x_1|f_2)$, $P(x_4)$, $P(x_3|x_1,x_4)$, and $P(x_2|f_1,x_1,x_4)$. The LSCDE training for each of these conditional densities with isotropic Gaussian kernels involves two hyperparameters λ and σ in addition to selecting a specified number of basis functions from the data. The first hyperparameter λ comes from equ 30. This equation is used to find the basis kernel function weights and λ acts as a regularization parameter. The other hyperparameter is σ and comes from the isotropic Gaussian kernel function, equ 42, where sigma acts as a bandwidth and controls how far basis function strength extends. For this simple test function, each node is assumed to be sufficiently modeled with 50 basis functions and a gird search is performed for the hyperparameters λ and σ. With the grid search, Sugiuama suggests using cross-validation and selecting the combination of hyperparameters that have the best Kullback-Leibler (KL) divergence[154].

Once the LSCDE training is complete for each node, the Bayesian network representation is complete. The resulting Bayesian network is then sampled using simple logic sampling because there is no evidence. The results of this sampling should follow the joint probability distribution of the original, effectively recreating the Pareto frontier, figure 39. In the figure the marginals are seen to compare well back to the input data set in figure 37.

However there is one significant issue. The values of f_1, f_2 and x_1 are all highly correlated in the training data set and along the Pareto frontier. The logic sampling shows that while the general trend was captured, the resulting distributions are not a crisp representation of the Pareto frontier. This means the Bayesian network was not able to capture the joint distribution with as much accuracy as needed for the product family design inferences.

Because of the failure of this Bayesian network on this simple problem, changes need to be performed before attempting the Bayesian approach on a more complicated design space. The Pareto frontier is an example of a lower dimensional surface embedded in a higher dimensional design space. A manifold can be thought of as a space that locally, but not globally, like a Euclidean space[96]. This means that the distance between two points is not what a Euclidean calculation would produce. Instead distance between two points in the manifold is called a geodesic where the distance is the shortest path between the two along the surface. The tangent vector along this manifold describes the local directions of the manifold.

Manifold learning is an area of much research in computer science and machine learning, producing many methods to extract the low dimensional structure in the high dimensional space. In manifold learning, statistical and heuristic techniques are typically applied in an attempt to answer questions about geometry from bulk data. There has been significant research resulting in a multitude of different techniques: Isomap, Local linear embedding, Laplacian Eigenmaps, Local Tangent Space Alignment, Hessian Eigenmaps, and Diffusion Maps[69]. Typically these algorithms begin by building a nearest neighbor network graph. This mesh-like structure then serves as a basis for the manifold structure. Geodesics can be approximated by the distance between two points along the graph. Once the graph is created, the various methods apply differing techniques to generate the embedding of the manifold points.

To apply this to the development of the Bayesian network approach, the embeddings and the ability to capture geodesics are not paramount. However, preserving the local manifold structure

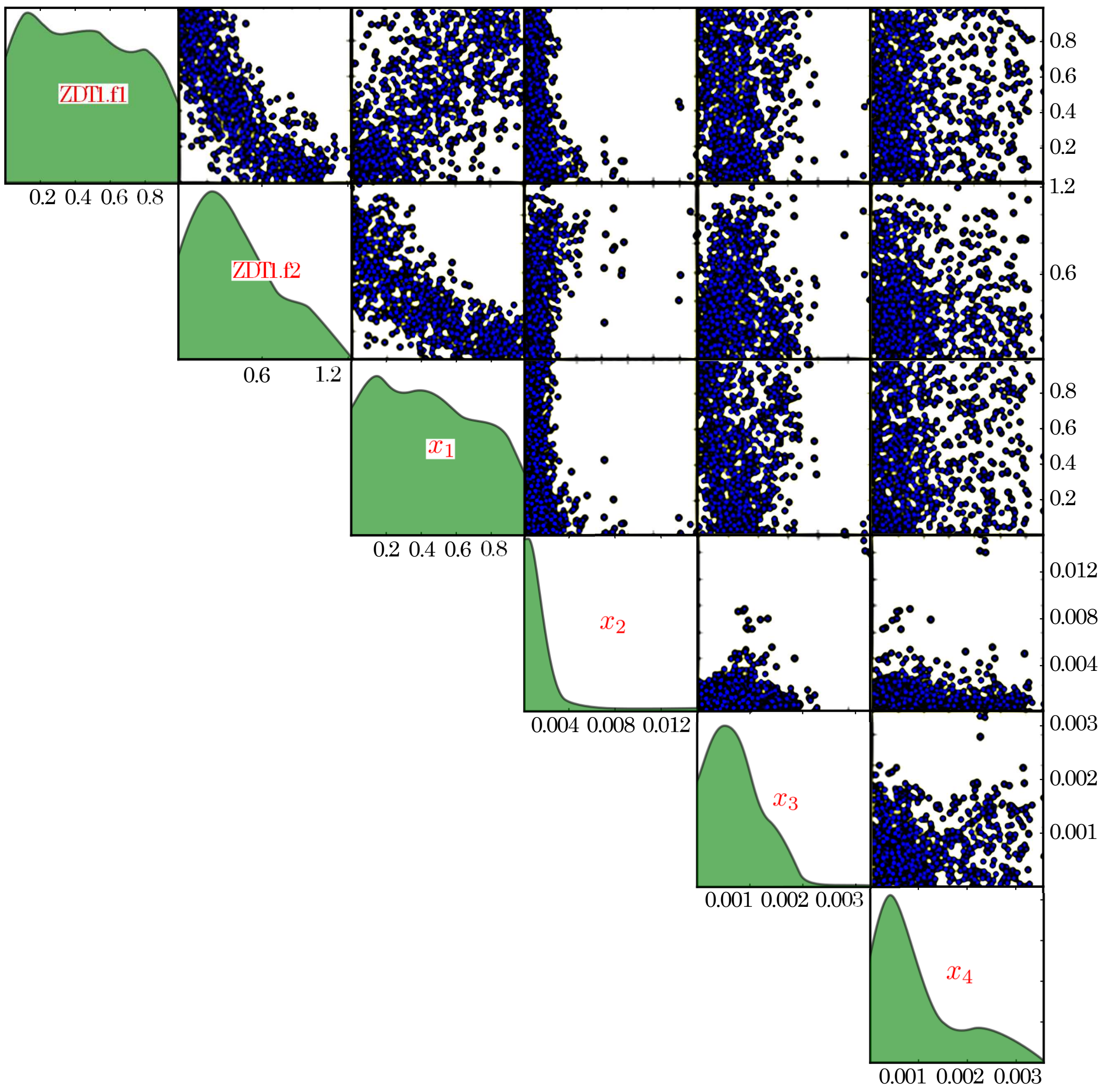


Figure 39: Isotropic Kernel Sample

is useful. One technique to accomplish this is to revisit the kernel function and attempt to use the anisotropic kernel function, equ 41. It is believed that the covariance matrix can incorporate local information to stretch the kernel strength along the local tangent space of the manifold[18].

Now the need is to calculate the covariance matrix. For the anisotropic Gaussian kernel functions, the covariance must be local for each basis function instead of using the global covariance for all of the data. One approach would be to begin in a similar manner to the manifold learning technique by building a nearest neighbor graph and using the closest k points to calculate the covariance. Similarly the covariance could be calculated by incorporating all of the points within a specified radius ε. However, both of these methods can encounter issues for data that is not evenly spaced across the entire manifold. Vincent proposes a more robust technique by weighting closest points more heavily for the covariance calculation[164]

$$C_{\phi_l} = \frac{\sum_{j=1,j\neq l}^{b} \phi\left(x_l,x_j\right)\left(x_l-x_j\right)\otimes\left(x_l-x_j\right)}{\sum_{j=1,j\neq l}^{b} \phi\left(x_l,x_j\right)} \tag{44}$$

where $\left(x_i-x_j\right)\otimes\left(x_i-x_j\right)$ is the outer product and $\phi\left(x_l,x_j\right)$ is a kernel function. If a uniform kernel is used this calculation becomes the global covariance. The evaluation of the anisotropic Gaussian kernel function requires the covariance to be inverted. This inverse may not exist for some data sets, like linear manifolds. Another factor δ introduced along the diagonal can make inverting the covariance possible

$$\Sigma_l = C_{\phi_l} + \delta^2 I \tag{45}$$

The original isotropic Gaussian kernels were selected because they could be analytically integrated which improves calculation efficiency. Using the anisotropic Gaussian kernel in the LSCDE fitting requires reevaluating the integrals in equ 33 and equ 35. Repeating equ 33

$$\begin{aligned} \Phi_{l,l'}\left(\bar{x}\right) &= \int \phi_l\left(\bar{x},y\right)\phi_{l'}\left(\bar{x},y\right)^T dy \\ &= \left(\sqrt{2}\sigma\right)\exp\left[-\frac{\xi_{l,l'}\left(\bar{x}\right)}{4\sigma^2}\right] \qquad (46) \\ \xi_{l,l'}\left(\bar{x}\right) &:= 2\left\|\bar{x}-\bar{u}_l\right\|^2 + 2\left\|\bar{x}-\bar{u}_{l'}\right\|^2 + \left\|v_l-v_{l'}\right\|^2 \qquad (47) \end{aligned}$$

And repeating equ 35

$$\int \hat{\alpha}^T \phi(\bar{x}, y)\, dy = \left(\sqrt{\pi}\sigma\right) \sum_{l=i}^{b} \hat{\alpha}_l \exp\left(-\frac{\left\|\bar{x}-\bar{u}_l\right\|^2}{2\sigma^2}\right) \tag{48}$$

To help simplify the integral problem some notation is introduced where $\begin{bmatrix} \bar{x} \\ y \end{bmatrix}$ is the point to evaluate and $\begin{bmatrix} u_l \\ v_l \end{bmatrix}$ is the basis l location so that $q_l = \begin{bmatrix} u_l \\ w_l \end{bmatrix} = \begin{bmatrix} \bar{x}-\bar{u}_l \\ y-v_l \end{bmatrix}$. Additionally the covariance matrix can be written as a block matrix $\Sigma_l^{-1} = M_l = \begin{bmatrix} U_{0_l} & V_l \\ V_l^T & W_{0_l} \end{bmatrix}$. Now equ 33 can be rewritten as

$$\Phi_{l,l'}(\mathbf{x}) = \int \exp\left[-\frac{1}{2}\left(q_l^T M_l q_l + q_{l'}^T M_{l'} q_{l'}\right)\right] dy \tag{49}$$

$$\begin{aligned} \Phi_{l,l'}(\mathbf{x}) = {} & \exp\left[-\frac{1}{2}\left((\bar{x}-\bar{u}_l)^T U_l (\bar{x}-\bar{u}_l) + (\bar{x}-\bar{u}_{l'})^T U_{l'} (\bar{x}-\bar{u}_{l'}) + N_{l,l'}^T S_{l,l'}^{-1} N_{l,l'}\right)\right] \\ & \cdot \int \exp\left[-\frac{1}{2}\frac{\left(y-\mu_{l,l'}\right)}{\sigma_{l,l'}^2}\right] dy \end{aligned} \tag{50}$$

where

$U_l = U_{0_l} - V_l W_{0_l}^{-1} V_l^T$

$\sigma_{l,l'}^2 = W_{0_l} + W_{0_{l'}},$

$\mu_{l,l'} = -W_{0_l}^{-1} V_l^T (\bar{x}-\bar{u}_l) - v_l + W_{0_{l'}}^{-1} V_{l'}^T (\bar{x}-\bar{u}_{l'}) - v_{l'},$

$S_{l,l'} = W_{0_l}^{-1} + W_{0_{l'}}^{-1}$

$N_{l,l'} = w_l - w_{l'}$

Finally because of the need to limit the design variable ranges, the integration limits introduce the error function, erf.

$$\int_{y_1}^{y_2} \exp\left[-\frac{1}{2}\frac{\left(y-\mu_{l,l'}\right)}{\sigma_{l,l'}^2}\right] dy = \sqrt{\frac{\pi}{2}}\sigma_{l,l'}\left[\operatorname{erf}\left(\frac{y_2-\mu_{l,l'}}{\sqrt{2}\sigma_{l,l'}}\right) - \operatorname{erf}\left(\frac{y_1-\mu_{l,l'}}{\sqrt{2}\sigma_{l,l'}}\right)\right] \tag{51}$$

The other integral, equ 35 is the integral to normalize the kernel density ratio estimate to make it a probability and becomes equ 52

$$\int_{y_1}^{y_2} \hat{\alpha}^T \phi(\bar{x}, y)\, dy = \sum_{l=i}^{b} \hat{\alpha}_l \sqrt{\frac{\pi}{2W_{0_l}}} \left[\mathrm{erf}\left(\frac{y_2 - \mu_l}{\sqrt{2W_{0_l}}} \right) - \mathrm{erf}\left(\frac{y_1 - \mu_l}{\sqrt{2W_{0_l}}} \right) \right] \cdot \exp\left[-\frac{1}{2} (\bar{x} - \bar{u}_l)^T U_l (\bar{x} - \bar{u}_l) \right] \tag{52}$$

where $\mu_l = v_l - W_{0_l}^{-1} V_l^T (\bar{x} - \bar{u}_l)$. Now the Bayesian network can be refitted using the anisotropic kernel function equ 41 and replacing the original integrals with equ 49 and equ 52. The base structure of the network does not change, however the conditional densities of the nodes are significantly more focused along the Pareto frontier. There are still two hyperparameters that are used to fit the densities, λ and σ. λ controls the regularization of the basis weights to help stabilize the calculation. However, the meaning of σ is different. Before, σ was the kernel bandwidth of the isotropic kernels and now is the isotropic Gaussian kernel bandwidth for calculating the covariance in equ 44. As with before, the best combination of λ and σ is selected using a grid search to determine the best KL-error using cross-validation of data. Figure 40 shows the logic sampling from the Bayesian network trained with the anisotropic kernel function. There has been a significant improvement with this networks ability to capture strong correlations among variables.

The LSCDE, modified to use the more flexible anisotropic kernel, is able to accurately capture the behavior of this simple Pareto frontier. Archambeau and Verleysen have another study using anisotropic kernels to output ordinary Gaussian mixtures when applied to manifolds[9]. They note the failure of ordinary Gaussian mixtures because of a discrepancy between the Euclidean space and the geodesic distance of the manifold. Nonstandard kernels allow for accurate depicting of manifolds[41]. This is because the probability density of the kernel needs to adequately reflect the local structure of the tangent space on the manifold. Bengio, Larochelle, and Vincent note that the prediction of the anisotropic kernel may not be as smooth as the underlying manifold[12]. Their recent work adds an additional assumption that the distribution in one area could be informative about the shape in a nonadjacent region to help predict where data may be sparse. However this is not necessarily a good assumption for a heteroscedastic SoS density approximation.

The final test with the ZDT1 function involves using the Bayesian network in a PMBGA. The goal is to verify that the Bayesian network continues to learn from previous generations and builds a model of the nondominated members of the population. If this is true, then the PMBGA will be

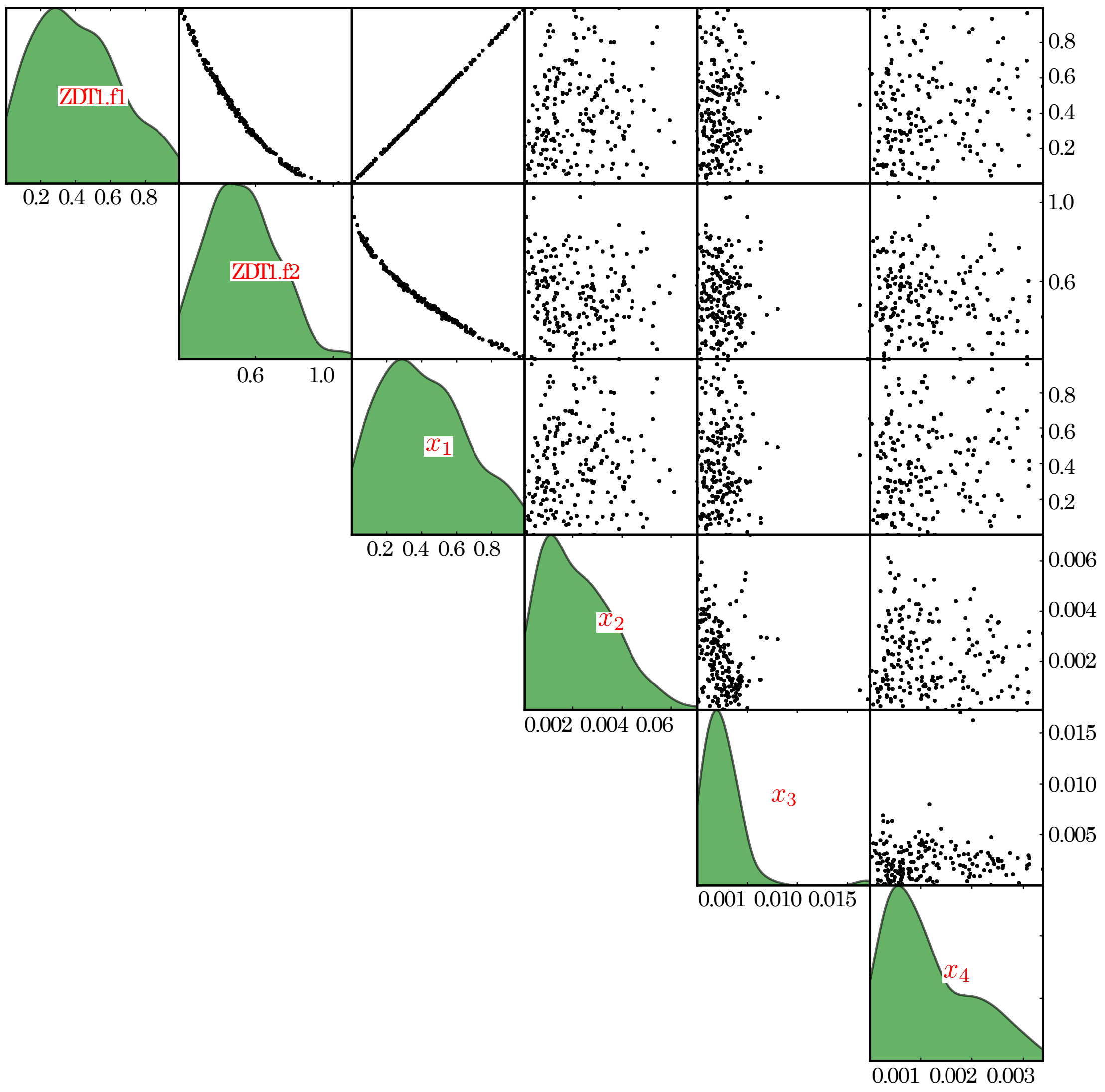


Figure 40: ZDT1 Sample from Bayesian Network using Anisotropic Kernels

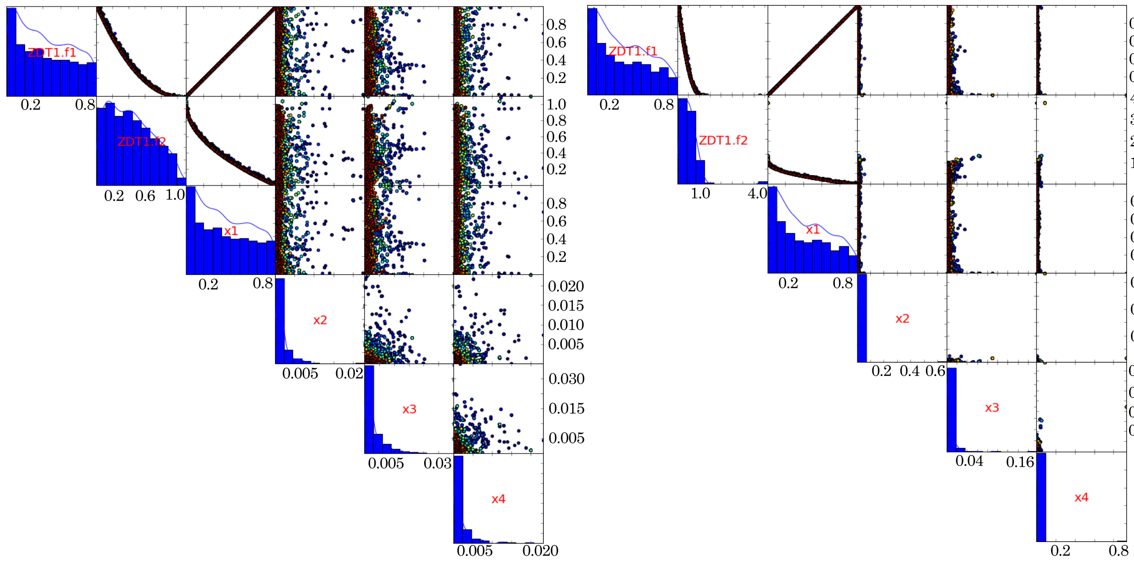


Figure 41: Comparison between NSGA-II (left) and PMBGA (right)

able to converge to the Pareto frontier. Figure 41 shows the final generation using the PMBGA. The biggest difference between the two is the PMBGA front found the max value of f_2 to be 4 compared to the NSGA-II max value of 1. This indicates the PMBGA may have prediction limitations near the edges of the space making f_2 converge more slowly. However, even though the PMBGA Pareto frontier is identical, the Bayesian network using the anisotropic Gaussian kernels has been able to converge to the majority of the behavior of the Pareto frontier. Now with the kernel selection finalized, one can use the Bayesian network to perform inference to test its prediction accuracy.

6.2 Bayesian Network Surrogate Model Example Application

The purpose of this section is to illustrate the ability of the Bayesian network to encode the joint probability distribution of a typical aerospace problem by using it to perform statistical inferences. If the network is able to accurately predict the output distributions for a simple problem then it supports its use in a large and more complicated product family design problem. To test the inference capability this Bayesian network will be trained to a design space exploration of the Breguet range equation. The Breguet range equation calculates the range an aircraft can fly as a function of several variables

$$R = \frac{V \cdot L/D}{TSFC} \ln\left(\frac{1}{1-\zeta}\right) \tag{53}$$

Table 6: DoE Ranges

	Lower Bound	Upper Bound
$V\ [km/h]$	350	450
$TSFC[kg/h/N]$	0.7	1.3
L/D	10	20
$frac$	.3	.8

where:

R - range

V - aircraft velocity

L/D - lift to drag ratio

$TSFC$ - Thrust specific fuel consumption

ζ - Fuel fraction

The equation combines elementary properties of the aircraft to predict range. These properties are aggregated characteristics of the aircraft. The $TSFC$ incorporates many low level properties of the aircraft's propulsion system, i.e. more efficient engines will increase the range of the aircraft. Lift to drag is representative of the aircraft's aerodynamic properties and geometry. The aircraft velocity is an operational mission parameter. Lastly, the fuel fraction is a relationship based on material properties, the weight breakdown of the aircraft to fuel, structure, and payload.

Although this is not the most complicated design code available, it is still reasonably challenging to visualize because of the five dimensions. Figure 42 shows a scatterplot matrix where some trends are easily visible. As expected given the domain of the variables, table 6, range is strongly correlated with the fuel fraction and to lesser extent L/D and TSFC.

The next step is to encode the joint probability distribution of this "high" dimensional point cloud with a Bayesian network. The first step is to determine the structure of the network. The converged-to network structure for the Breguet range equation is shown in figure 43. The structure confirms the interactions between the variables. None of the variables are conditionally independent. There is only one degree of freedom and all but one of the variables must be specified to determine the last.

After the LSCDE has found an estimate for the nodes, the network can be used for inference.

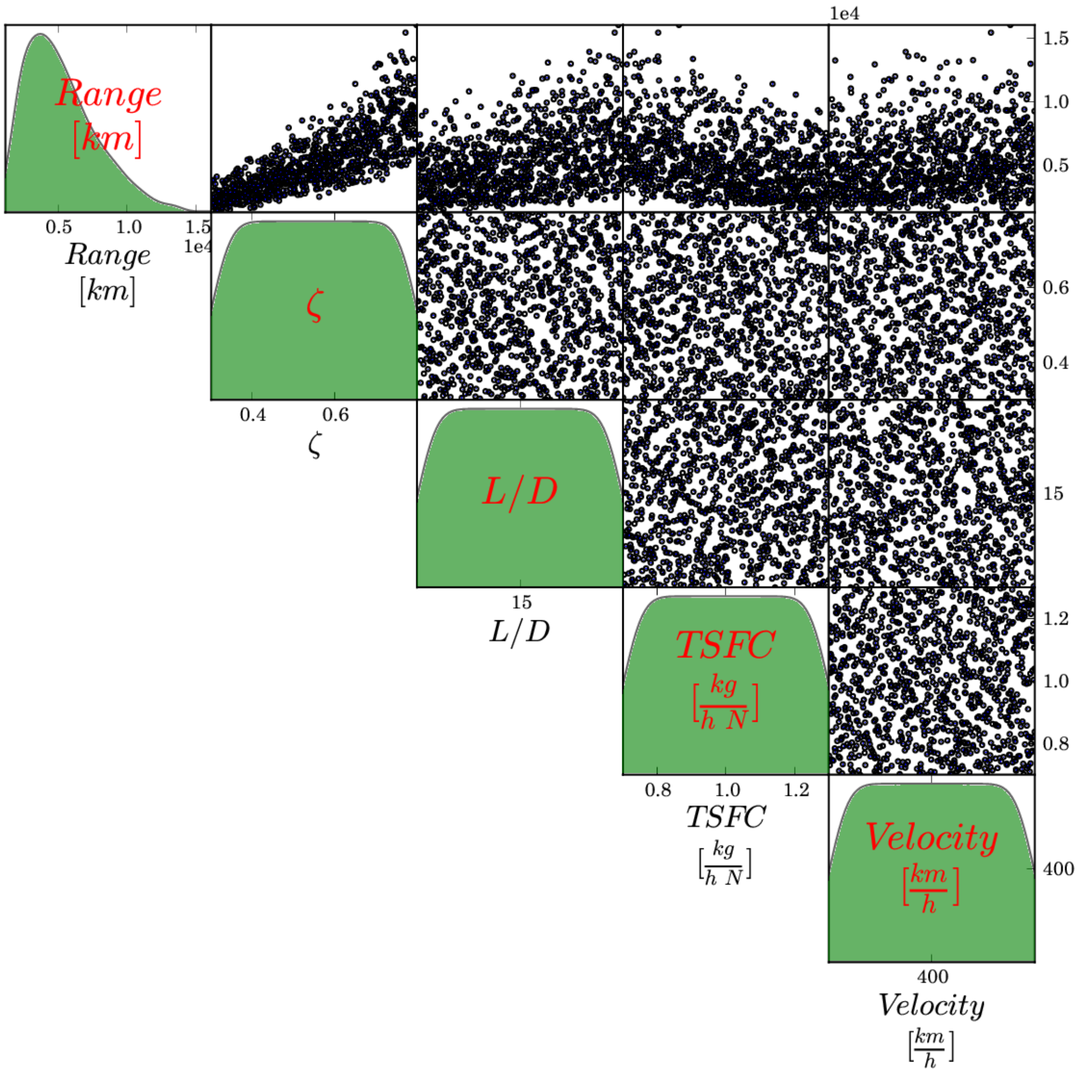


Figure 42: Breguet Range Equation DOE

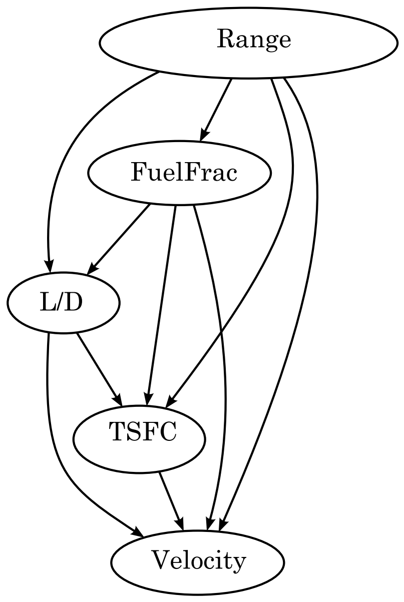


Figure 43: Bayesian Network Structure for the Breguet Range Equation DOE

The easiest test for the network is to verify that the network has properly captured the joint distribution of the training set. For this test, simple logic sampling is performed because there are no "observed" evidence nodes. The first step in logic sampling is to arrange the nodes in topological order. In this case

1. Marginal of Range - $P(Range)$
2. Conditional of Fuel fraction - $P(\zeta|Range)$
3. Conditional of L/D - $P(L/D|Range,\zeta)$
4. Conditional of TSFC - $P(TSFC|Range,\zeta,L/D)$
5. Conditional of Velocity - $P(Velocity|Range,\zeta,L/D,TSFC)$

These conditional distributions are then sampled in order until the number of desired samples are generated. Figure 44 shows the results of 2,000 samples from the Bayesian network. Ideally, if the network has properly captured the joint distribution, this figure should compare well to the scatterplot matrix from the DOE in figure 42. Indeed the general magnitudes of the variables are the same, the marginal distributions are roughly equivalent, and the strong trends are the same as well.

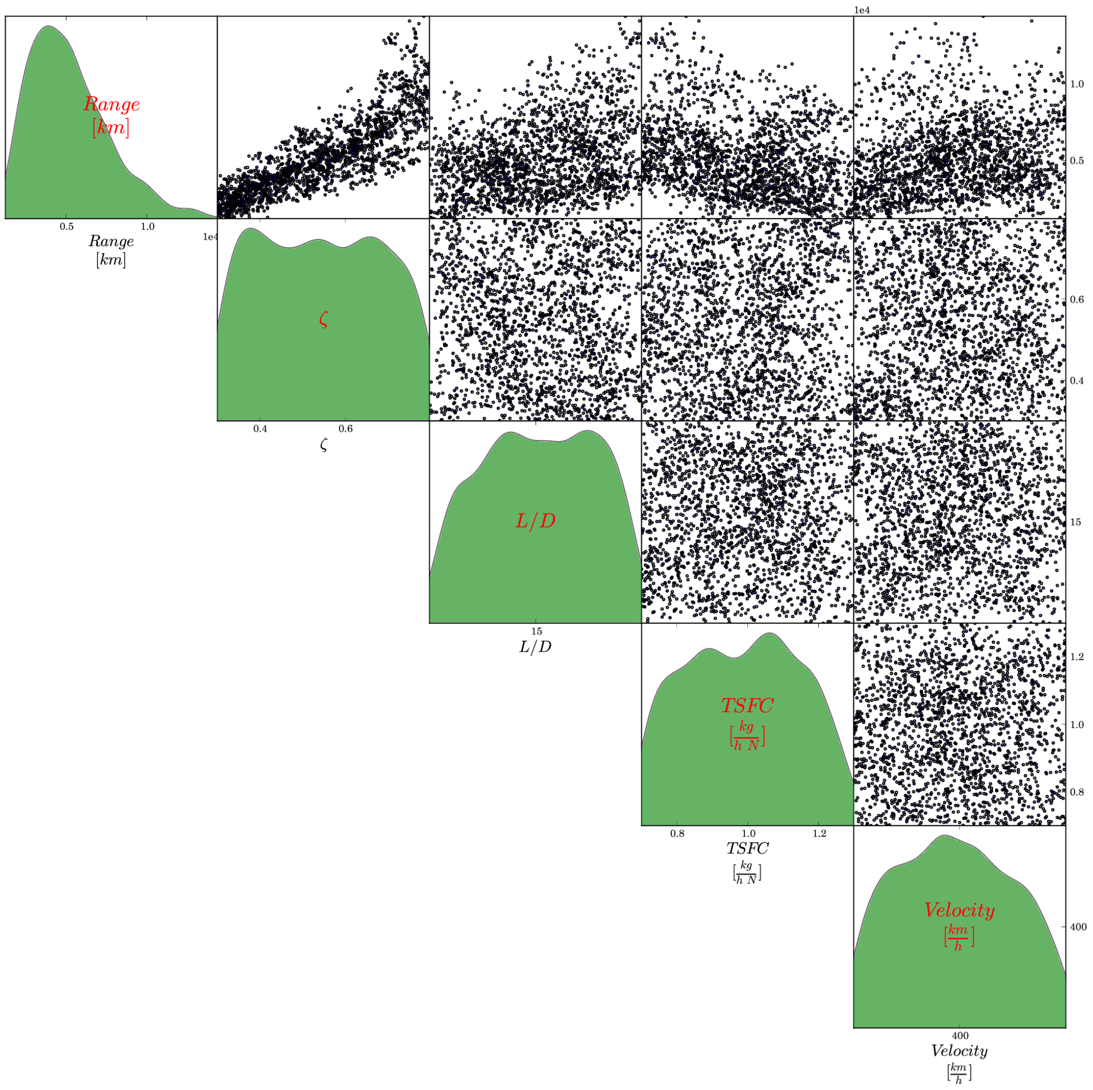


Figure 44: Joint Probability Distribution Sample of the Bayesian Network

Upon deeper inspection, the scatterplot alone does not clearly show all of the structure of the relationships that exist between the different variables. Perhaps a better visual indication of the quality of the Bayesian network comes from the use of parallel axis plots. Figure 45 gives a side by side comparison of the variable interactions. The top parallel axis plot uses the DoE training data and the bottom uses the 2,000 samples from the network. Both parallel axis plots have the points colored by the range. This figure is much more revealing of the internal structure that has been captured in the Bayesian network.

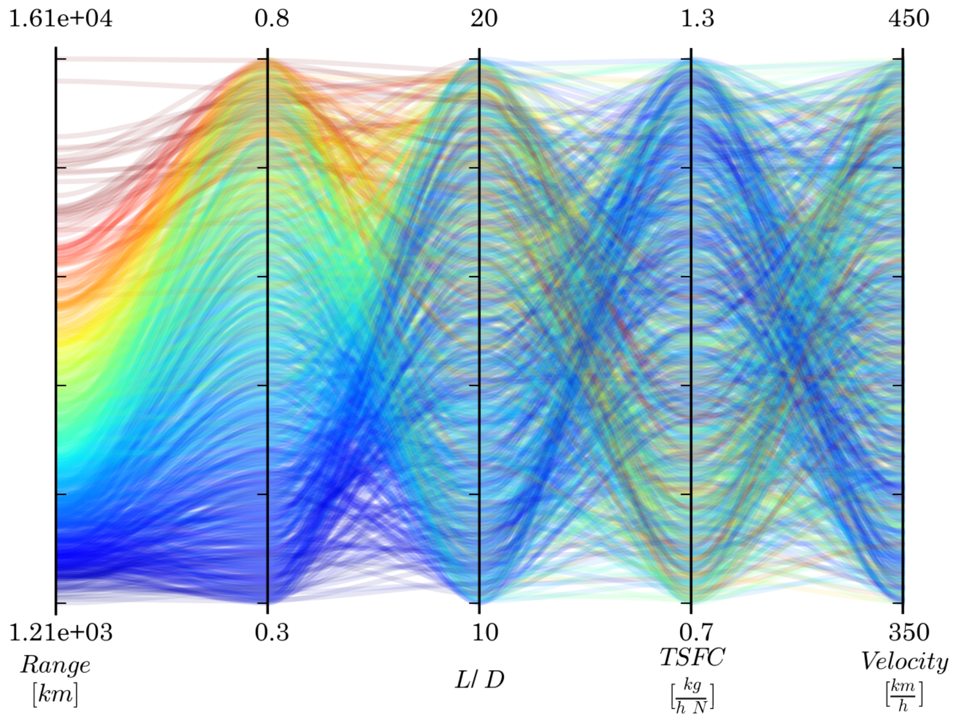
Now that the network is shown to encode the variable interactions, additional tests can be performed to gain addition confidence into the feasibility of the Bayesian network as a probabilistic surrogate model. To demonstrate the accuracy and power of Bayesian inference using importance sampling, one can pose the following question. What is the likely aircraft range given the aircraft has a fuel fraction of .5, L/D of 17, TSFC of $1[kg/h/N]$ and flies at a velocity of $400[km/h]$? From the Breguet range equation equ 53, this can be calculated directly:

$$Breguet.Range\left(FuelFrac = .5, LD = 17, Velocity = 400, TSFC = 1[kg/h/N]\right) = 4713.4km$$

However the purpose is to test the trained network. So this question is mathematically formulated as:

$$P\left(Breguet.Range | FuelFrac = .5, LD = 17, Velocity = 400km/h, TSFC = 1[kg/h/N]\right)$$

Because of the evidence nodes the inference is performed on the Bayesian network using importance sampling. This evidence then is assumed to be uniformly distributed around the target values mentioned above with a tight tolerance. When executing the importance sampling the evidence nodes are sampled using the assumed distributions while any unobserved nodes are sampled once all other parents have been sampled. This requires the samples to be weighted to correct for unlikely values from the evidence. However this method remains significantly faster than pure logic sampling in the presence of evidence. Figure 46 shows the resulting posterior distribution of this weighted sample. Each variable's expected value is recorded as well as the evidence distributions



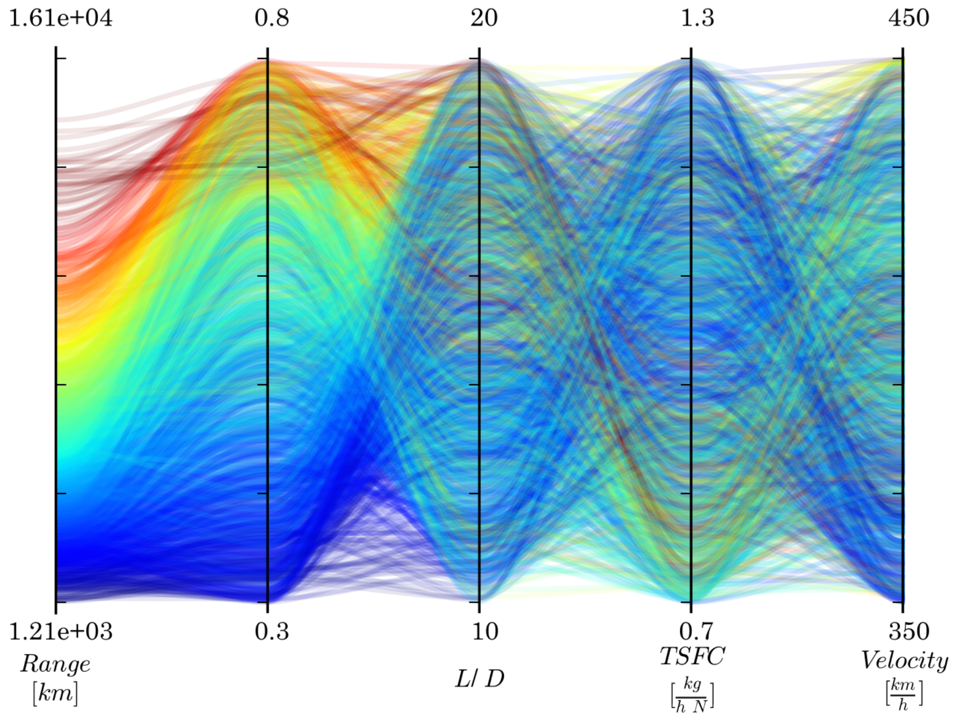


Figure 45: Training Data (top) Sampled Data (bottom)

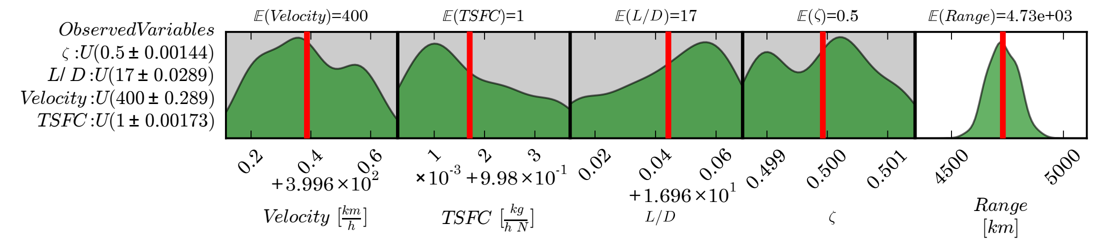


Figure 46: $P\,(Breguet.Range|FuelFrac = .5, LD = 17, Velocity = 400, TSFC = 1)$

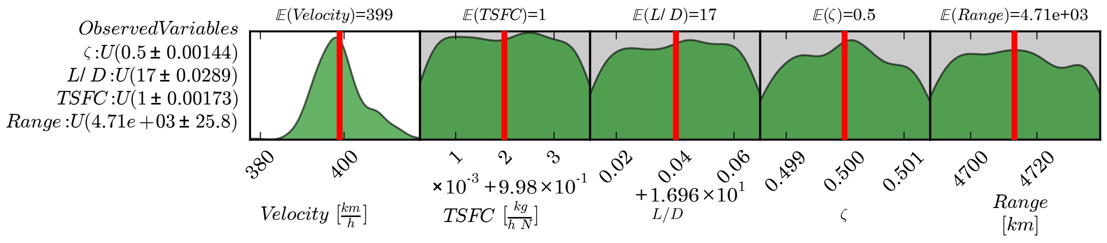


Figure 47: $P\,(Velocity|Breguet.Range = 4713, FuelRac = .5, LD = 17, TSFC = 1)$

and bounds. Using the range equation the value is 4713 km and the results of the inference show an expected value for the range to be 4,729.4 km with a standard deviation of 194.6 km.

The power of the Bayesian Network inference comes from its ability to change the form of the question. For example, what is/should be the velocity:

$$P\,(Velocity|Breguet.Range = 4713, FuelFrac = .5, LD = 17, TSFC = 1)$$

From before, the Velocity is known to be be 400 [km/h]. In general, to modify the inputs and outputs of the Breguet range equation would either require an algebraic manipulation to rearrange the terms or an iterative solver. However for importance sampling, the procedure remains the same as before with the exception that the evidence nodes include range and velocity is now unobserved. The posterior distribution of this importance sampling is shown in figure 47. The expected value now for the aircraft's velocity is 398.9 km/h with a standard deviation of 5.89 km/h.

Now that the network appears to have successfully captured the joint distribution and importance sampling inference returns reasonable values the next test is to predict a variety of points. There are typically two measures in goodness of fit tests, the model fit error (MFE) and the model representation error (MRE). With the MFE the model is tested to see how well it predicts the training data. Similarly the MRE tests the predicted model against data the model was not trained against.

To calculate the model prediction value for a set of data:

$$Data = \begin{pmatrix} e_{11} & \dots & e_{1m} \\ \dots & e_{ij} & \dots \\ e_{1n} & \dots & e_{mn} \end{pmatrix}$$

The predicted values are

$$Predicted = x_{i,Predicted} \forall i \in \{1, \dots, m\}$$

where the inferences are performed many times with the variable to be predicted being omitted from the evidence.

$$\begin{aligned} x_{i,Predicted} &= \left\{ \mathbb{E}\left(x_i | x_{kj} = e_{ij}\right) \, \forall j \in \{1, \dots, n\} \; i \neq k \right\} \\ x_{i,Actual} &= \left\{ e_{ij} \, \forall j \in \{1, \dots, n\} \right\} \end{aligned}$$

This process relies on taking the expected value of the posterior and fails for multimodal distributions. Fortunately this Breguet range equation is simple. Figure 48 shows the actual by predicted plots for all of the variables. Overall it reveals the model to be reasonably fit with the exception that high range values have larger residuals compared to low range values.

Because the inference sample requires sampling, the results are sensitive to the number of samples. Figure 50 shows a convergence test for several different range settings. One problem is the high range setting has a converged value with a greater error than the others.

Figure 50 shows that there are relatively few design cases that have an aircraft range greater than 120,000 km. This adds further evidence that the Bayesian network will have difficulties near the edges of the design space or where there are fewer design cases. In reality this is not unexpected behavior. Sparse regions of the design space are going to carry less penalty for poorer fits. In training the network, there are two hyperparameters that are found using a grid search to sweep over a range of possible values. The combination of hyperparameters that yields the best score are

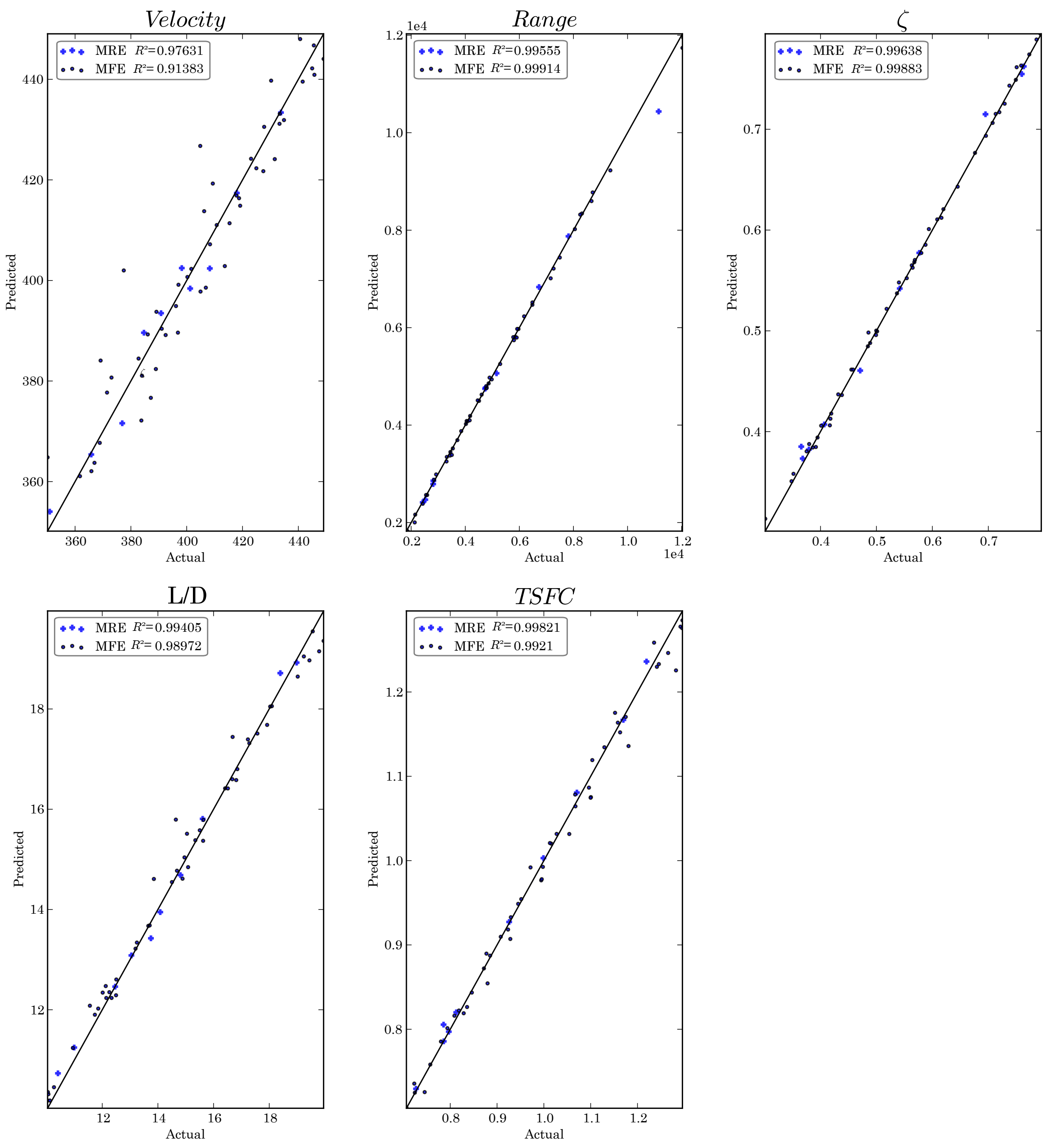


Figure 48: Breguet Goodness of Fit

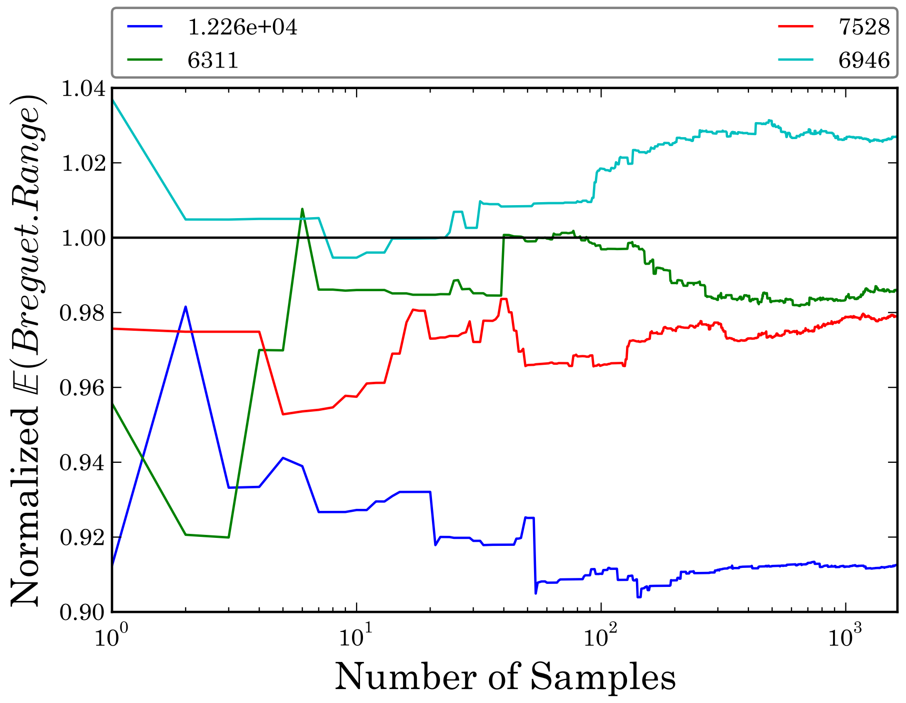


Figure 49: Breguet Expected Value of Range Convergence for Various Ranges

selected which means the majority of points well fit but low density regions may appear as outliers and not be as accurately represented. Additionally, the covariance calculation for the edges of the design space are going to have a more difficult time capturing the tangent space compared to regions that are evenly covered in all directions.

6.3 Implementation

This section discuses the implementation in more detail, specifically the analysis for the Bayesian network in Python. Python is a high level programming language with dynamic typing which allows for rapid generating and prototyping of a code. Combined with quality development environments and a growing community base makes Python a healthy ecosystem for scientific computing. Additionally, Python has many well developed and maintained scientific libraries: SciPy[74], NumPy[113].

Another promising Python library for scientific computing and engineering analysis is OpenMDAO[62]. OpenMDAO is a project currently under development at NASA Glenn Research Center. OpenMDAO is a flexible framework for multidisciplinary design analysis and optimization methods. It

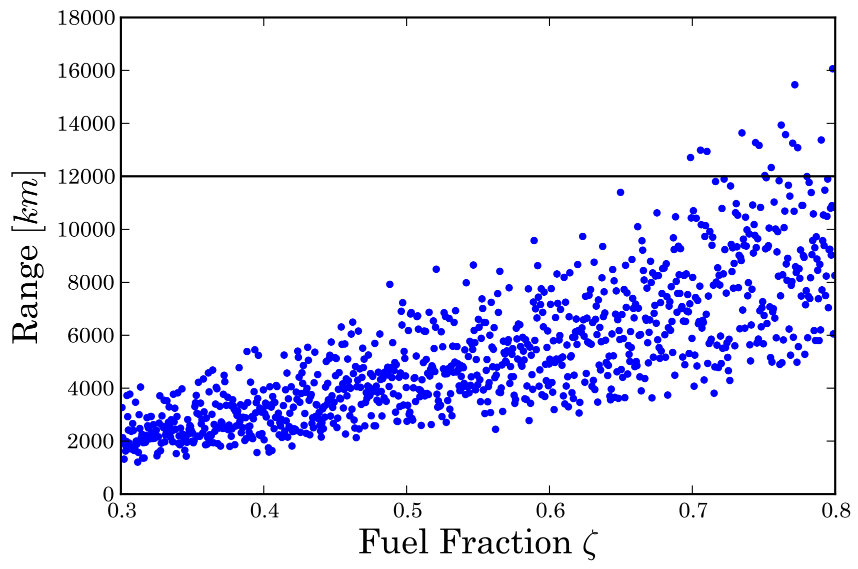


Figure 50: Breguet Range Sparsity

allows for designers to integrate many disciplinary design codes into a more complex environment. The library is still being actively developed but has much of the needed capability already implemented. Currently it is capable of switching driving algorithms like DoEs and optimizers without modifying design codes. OpenMDAO also has several preexisting extensions to other Python libraries for optimization in addition to native implementations.

Another key reason for developing in Python is the Python Environment for Bayesian Learning (Pebl). This open source library was designed for learning Bayesian network structure from data and prior knowledge[138]. Pebl is published under the MIT license with the desire for additional academic research into Bayesian network algorithms. Unfortunately it was last updated in 2011 and only handled discrete variables requiring the discretization of continuous domains and possible space state explosion. Additionally, Pebl does not natively have any inference or sampling capabilities. However Pebl was a useful starting point for this research.

The first step in the implementation was to fit the continuous densities using LSCDE. As explained previously, the fitting of the LSCDE requires a grid search over two hyperparameters. Significant development effort was on computational speed issues encountered as many repetitions are required before an accurate fit is determined. Part of this code optimization involved converting expensive function calls into Cython. Cython is a compilable language that understands Python syntax and enables near C speeds. Combined with Gaussian mixture sampling described below, this allows the improved Pebl library to operate on continuous domains and has reasonable execution times on networks with at least 30 to 40 variables.

The last key to reasonable inference times is being able to efficiently generate points from the LSCDE. The LSCDE models the density ratio using a mixture of kernel functions and in this case, anisotropic Gaussian kernels.

$$\begin{aligned} P(y|\bar{x}) &= \alpha^T \phi\left(\bar{x}, y\right) \\ &= \alpha_i \sum_{i=1}^{n} \frac{1}{2\pi \det\left(\Sigma\right)^{1/2}} \exp\left(-\frac{1}{2}\begin{bmatrix} y - v_i \\ \bar{x} - \bar{u}_i \end{bmatrix}^T \Sigma^{-1} \begin{bmatrix} y - v_i \\ \bar{x} - \bar{u}_i \end{bmatrix}\right) \end{aligned} \tag{54}$$

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \tag{55}$$

The density ratio can be thought of as a weighted sum of univariate Gaussian distributions that slice through the multivariate density function and some value $\bar{x}$ using manipulations similar to the analytical evaluation for the LSCDE integrals[140][123]

$$P(y|\bar{x} = \tilde{x}) = \sum_{i=1}^{n} w_i \mathcal{N}\left(\mu_i, \sigma_i\right) \tag{56}$$

where:

$$\begin{aligned} w_i^{!} &= \alpha_i \exp\left(-\tfrac{1}{2}\left(\tilde{x} - \bar{u}_i\right)^T \left[\Sigma_{22}\right]^{-1}\left(\tilde{x} - \bar{u}_i\right)\right) && (57) \\ \mu_i &= x_i + \Sigma_{12}\Sigma_{22}^{-1}\left(\tilde{x} - \bar{u}_i\right) && (58) \\ \sigma_i &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} && (59) \end{aligned}$$

Finally the basis function weights can be normalized.

$$w_i = \frac{w_i^{!}}{\sum_{i=1}^{n} w_i^{!}} \tag{60}$$

Depending on the number of samples needed, the active basis functions can be determined by sampling from a multinomial distribution with the given basis weights. Now that the active basis functions have been determined, the process of sampling from the conditional density for node has now simplified to sampling from a normal distribution. The last issue to overcome is that normal distributions are unbounded but it is necessary that the Bayesian network simulate variables same within the design variable ranges.

While sampling from the unbounded Gaussian mixture in the form above is sufficient for most variables there are times when a basis function is selected near the bounds of the design space. The naive solution is to resample the basis function and reject all that are outside the bounds. This can cause a sampling slowdown as many samples are generated that would need to be discarded before one is found inside the design space. Robert provides a remedy for this with an improved algorithm for sampling from truncated normal variables[127]. The normal distribution bounded on $[\mu^-, \mu^+]$ can be described by:

$$f\left(x|\mu,\mu^-,\mu^+,\sigma\right) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sqrt{2\pi}\left[\operatorname{erf}\left(\left(\mu^+-\mu\right)/\sigma\right)-\operatorname{erf}\left(\left(\mu^--\mu\right)/\sigma\right)\right]} \tag{61}$$

The process for simulated samples from this distribution is

Algorithm 4 Truncated Normal Sampling

```
for each sample from the active basis function do
    Sample $z\ U[mu_l, mu_u]$ {Generate a value uniformly between the lower and upper bounds}
    $\rho(z) = \begin{cases} e^{-z^2/2} & if\, 0 \in [\mu^-, \mu^+] \\ e^{\left(\left(\mu^+\right)^2 - z^2\right)/2} & if\, \mu^+ < 0 \\ e^{\left(\left(\mu^-\right)^2 - z^2\right)/2} & if\, 0 < \mu^- \end{cases}$
    $u \leftarrow U[0,1]$ {Accept Rate}
    if $u <= p(z)$ then
        x=z {Accept the value}
    end if
end for
```

With the implementation of truncated normal sampling on top of Gaussian mixture sampling it becomes computationally efficient to simulate large samples from the condition density estimates of the continuous nodes in the Bayesian network.

6.4 Conclusions

This chapter expanded the description of the Bayesian network needed for the probabilistic approach for identifying product family commonality. One of the primary goals of this chapter was to gain confidence in the ability for the Bayesian network to represent the joint distribution of continuous design spaces. To test its accuracy, two problems were used.

The first tested the implications of the kernel function used to estimate the conditional densities. It was found that for the Bayesian network to represent Pareto frontiers, the kernel function needs to be able to concentrate its predictive power along the local manifold space. This Pareto frontier test problem also was useful for establishing the ability of the Bayesian network to be used inside a PMBGA to converge to the Pareto frontier.

The second test problem focused on the Bayesian networks ability to represent a design space exploration of an elementary equation exemplifying aerospace problems. This test problem also served as a testbed for understanding Bayesian inference and determining its accuracy and applicability.

These two test problems have shown the Bayesian network to be a promising technique for capturing Pareto frontiers. Furthermore, the flexibility provided by the importance sampling allows for inference that may be useful in inverse design problems. Forward design is the typical analysis of components. For example, given a known set of design variables that describe a system what is that system's performance. In inverse design the goal is to start with a desired performance capability and arrive at a system; graphically this is shown in figure 51[15]. Currently this is frequently accomplished building surrogate models that allow more design cases to be executed. Bayesian inference enables a robust inverse design.

The third goal of this chapter was to discuss briefly the implementation of the Python library and the lower level sampling needed to be able to efficiently draw samples from the node conditional density estimates. This efficiency is critical to the use of Bayesian network modeling on higher dimensional problems to prevent runtimes from becoming prohibitively large. Now that the Bayesian

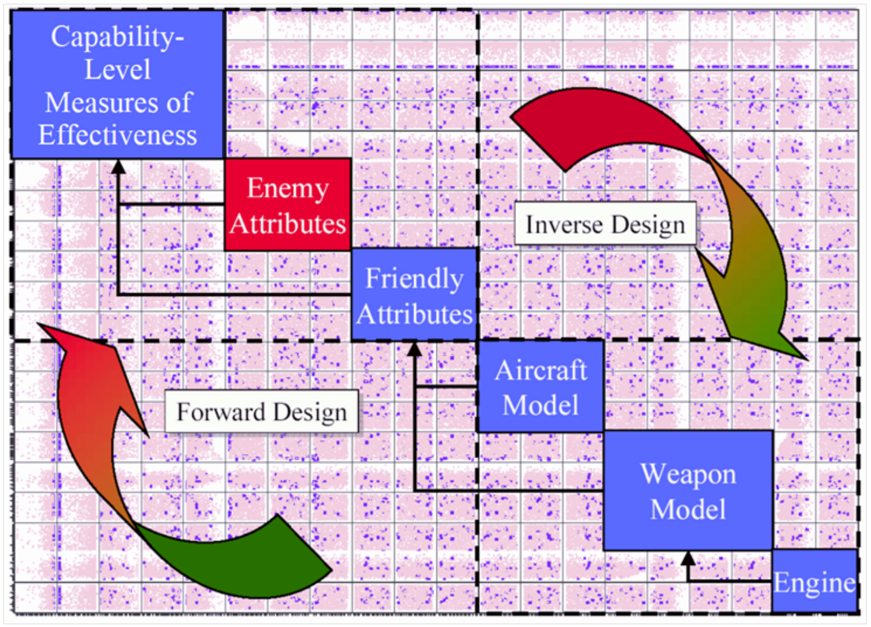


Figure 51: Multivariate View of the Top-Down Decision Making Process for Inverse Design [15]

network has been shown to work in principle on small test problems it can now be used in the complicated high dimensional product family experiment in chapter 7 and in the demonstration problem in chapter 8. The next chapter will now elaborate on the generic product family methodology by identifying component commonality.

Chapter VII

DESIGN OF AN ELECTRIC MOTOR FAMILY

This chapter presents a series of experiments testing the two proposed commonality identification approaches on a product family of universal electric motors. The electric motor example was introduced by Simpson where a detailed description of the theory behind the analysis model can be found[148]. This example product family design problem has since become a common test case for benchmarking product family methods[65, 107, 148, 98, 99]. The electric motor problem is a useful benchmark because it is a challenging Class III product family problem. The product requirements and analysis model are already specified. The objective of the problem is to design a product family of ten universal electric motors where each motor targets unique torque requirements, while maximizing the common components shared across the different motors. The experiments in this chapter allow for some conclusions to be drawn about the two proposed commonality identification approaches' effectiveness.

7.1 Motor Model

The universal electric motor model is described by a set of equations defining the relationships between the input and output parameters. These equations can be manipulated to solve for different variables. The model implementation presented here is from Hernandez[65]. This implementation was selected because the motor equations have been manipulated to have the target power and torque included as inputs and iteratively solves for the current, and stack length. This then allows for fast model execution because the torque and power are known for each motor.

The diagram of the universal electric motor is found in figure 52, as well as the input and output variables for this implementation. The relevant electric motor nomenclature is further described in table 7 with symbols, descriptions, and units.

The electric motor implementation takes the following variables as inputs $\left\{N_c, N_s, A_{wa}, A_{wf}, r, t\right\}$. In addition to these variables each electric motor product in the family has a target power output and unique torque target. There are also a few physical constants, table 8, that the motor model requires.

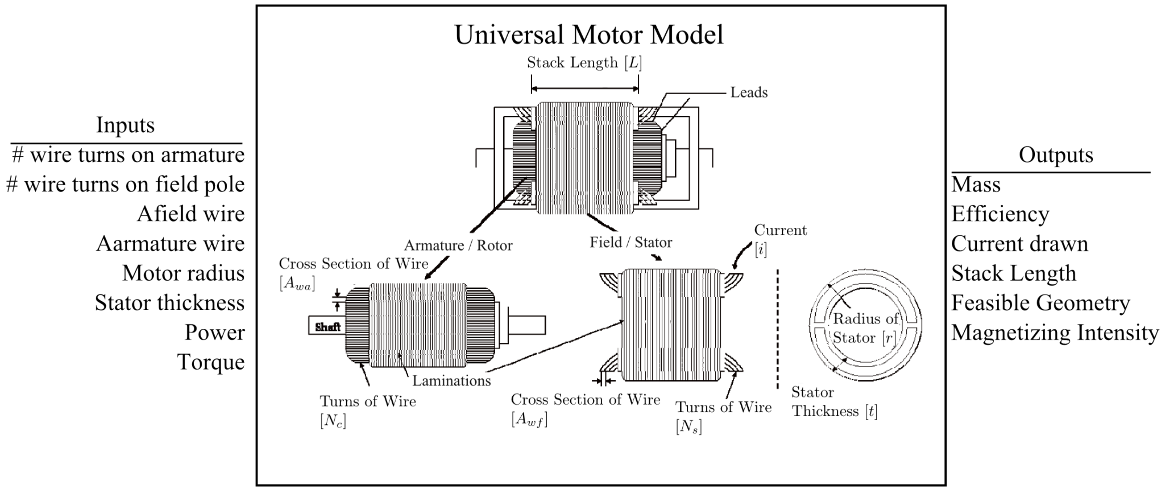


Figure 52: Universal Motor Model [148] [20]

Table 7: Universal Motor Nomenclature

Symbol	**Description**	**Units**
N_c	number of wire turns on the armature	
N_s	number of wire turns on the field pole	
A_{wa}	cross sectional area of the armature wire	mm^2
A_{wf}	cross sectional area of the field wire	mm^2
r	radius of the motor	m
t	thickness of the stator	m
i	current draw of the motor	Amp
L	stack length	m
P	power	W
T	torque	Nm
M	mass	kg
η	efficiency	

Table 8: Constants used in Universal Motor Model

Symbol	**Description**	**Value**
l_{gap}	Gap between rotor and stator	0.7 mm
N_p	Number of poles	2
V_t	Voltage	115 Volts
α		2 Volts
μ_a	Relative permeability of air	1
μ_o	Permeability of free space	$4\pi \times 10^{-7}$ Henry/m
ρ	Resistivity of copper	1.68×10^{-8} Ohm $\cdot$ m
ρ_{copper}	Density of copper	8960 kg/m^3
ρ_{steel}	Density of steel	7850 kg/m^3

The analysis of an electric motor begins by calculating the values of C, D, E, and F which are given in Eqs 62 - 65:

$$\begin{aligned}
C &= \frac{N_c N_s}{\pi}\mu_o\left[\frac{l_c}{2\cdot t\cdot\mu_s}+\frac{1}{\mu_s}+\frac{l_{gap}}{l_r\cdot\mu_a}\right]^{-1} && (62)\\
D &= V_t-\alpha && (63)\\
E &= -4\rho\left[(r-t)\left(\frac{N_c}{A_{wa}}+\frac{N_p N_s}{A_{wf}}\right)-\frac{l_{gap}N_c}{A_{wa}}\right] && (64)\\
F &= -2\rho\left[\frac{N_c}{A_{wa}}+\frac{N_p N_s}{A_{wf}}\right] && (65)
\end{aligned}$$

Where μ_s is relative permeability of steel and depends on magnetizing intensity, H:

$$\mu_s=\begin{cases}0.2279H^2+52.411H+3115.8 & H\le 220\\ 11633.5-1486.33\cdot\ln(H) & 220<H\le 1000\\ 1000 & H>1000\end{cases} \tag{66}$$

Equ 67 allows the calculation of the magnetizing intensity:

$$H=\frac{N_c i}{l_s+d_a+2l_{gap}} \tag{67}$$

Where l_s and d_a are the mean path length within the stator and the diameter of armature respectively and are determined by:

$$\begin{aligned}
l_c &= \frac{\pi}{2}(2r+t) && (68)\\
d_a &= 2(r-t-l_{gap}) && (69)
\end{aligned}$$

In the original electric motor studies, a programming error doubled the output of the magnetizing intensity, equ 67. Because the purpose of including the electric motor is to benchmark against other research, the error is maintained for this analysis. Also note that equ 67 depends on the value of the motor current, i. This implementation of the electric motor problem has the torque and power as input parameters so the equ 70 and equ 71 can be rearranged, to solve for i as a function of torque and power, equ 72.

$$\begin{aligned} T &= C \cdot i^2 L && (70) \\ P &= D \cdot i + E \cdot i^2 + F \cdot i^2 L && (71) \end{aligned}$$

$$i(T,P) = \frac{-CD + \sqrt{C^2D^2 + 4C^2EP - 4CEFT}}{2CE} \quad (72)$$

Equ 72 requires an iterative process due to the circular dependence between C and i. This is accomplished by iterating equ 62 through equ 72 to converge to the correct value of i. After converging, the stack length of the motor can be determined, equ 73:

$$L(T,P) = \frac{T}{C \cdot [i(T,P)]^2} \quad (73)$$

Equ 73 also indicates that the stack length is directly proportional to the motor's torque which is the main product differentiator in this study. Finally, the mass, equ 74, and efficiency, equ 75, of the motor can be calculated:

$$\begin{aligned} m &= A + B \cdot L && (74) \\ \mu &= \frac{D + E \cdot i + F \cdot i \cdot L}{D} && (75) \end{aligned}$$

Where:

$$\begin{aligned} A &= 4\rho_{copper}\left[(r-t)\left(N_cA_{wa} + 2N_sA_{wf}\right) - l_{gap}N_cA_{wa}\right] && (76) \\ B &= \pi\rho_{steel}\left[r^2 - 2(r-t)\,l_{gap} + l_{gap}^2\right] && (77) \end{aligned}$$

Chamberlain offers several observations about the electric motor design variables and their importance in family commonality[20]. For instance, the number of wire turns on the field or the armature are integer values, but there may be little value to having the number of turns be common within the family since the manufacturing process to wind the coils is easily changed. On the other hand, the gauge of the wires A_{wa} and A_{wf} would be beneficial to share, because each time the gauge

Table 9: Universal Motor Variable Bounds

Symbol	Description	Lower Bound	Upper Bound	Units
N_c	Number of wire turns on the armature	100	1500	
N_s	Number of wire turns on the field pole	1	500	
A_{wa}	Cross-sectional area of the armature wire	0.01	1.0	mm^2
A_{wf}	Cross-sectional area of the field wire	0.01	1.0	mm^2
r	Radius of the motor	0.01	0.1	m
t	Thickness of the stator	0.0005	0.01	m
i	Current draw of the motor	0.1	6	Amp
L	Stack length	0.001	.1	m

changes, the winding machines need to be stopped to change spools, which slows down manufacturing. Also, if the family requires a number of different wire gauges, then they must be stored in a warehouse, adding to operating cost. The r and t are stator dimensions and commonality could be considered valuable. Additionally the stack length, L, has a strong influence on the family and would require many smaller design changes if it were to change. Lastly, the current is external to motor and it not really considered as a platform variable.

7.2 Product Family Design Problem

Now that the electric motor model has been described, the family design problem can be posed. First, table 9 gives the ranges for the relevant design parameters, which are consistent with other product family design studies.

In the formulation of the product family design problem, the current drawn is a state variable, adjusted, so the output power matches the customer requirements. There are seven physical design parameters to consider as platform modules, $\{N_c, N_s, A_{wa}, A_{wf}, r, t, L\}$. As there are ten different motors being considered, there are a total of $115,975$ different sharing combinations of any particular module. If we consider each of the design parameters as describing its own module space, then there are a total of $115,975^7$ different platform configurations for the family.

Formally, the electric motor family design problem can be stated as minimizing the mass, and maximizing the efficiency for each of the ten motors in the family while trying to maximize the commonality in the family. The optimization is performed with respect to six design variables (note stack length, L, is determined for each motor) and is subject to several constraints. In addition to the variable bounds, each motor has a specified target torque, T, while maintaining an output power of 300 Watts. Furthermore, if the magnetizing intensity is too high or the radius is greater than the thickness, the motor is infeasible.

$$\text{maximize}: \begin{cases} -M^p & \text{mass} \\ \eta^p & \text{efficiency} \\ \sum_{ijpq} R_{ij}^{pq} & \text{commonality} \end{cases} \qquad p = \{1,2,..,10\}$$

$$\text{with respect to:} \quad \begin{array}{ll} \{N_c, N_s, A_{wa}, A_{wf}, r, t\}^p & \text{design variables} \\ R & \text{commonality relations} \end{array}$$

subject to: Variable Bounds Table 9

$$T^p = \begin{array}{c} \{0.05, 0.1, 0.125, 0.15, 0.2, \\ 0.25, 0.3, 0.35, 0.4, 0.5\} \end{array} \text{Nm}$$

$$P = 300\,\text{Watts}$$

$$H \leq 5{,}000\,\text{Amp} \cdot \text{turns/m}$$

$$\eta \geq 15\%$$

$$r > t$$

$$M \leq 2\,\text{kg}$$

$$R_{ij}^{pq}\left(x_i^p - x_j^q\right) = 0 \qquad \text{for } p, q \in \mathcal{P};\ p < q$$

$$R_{ij}^{pq} \in \{0,1\}$$

$$(i,j) \in \mathbb{S}^{pq}$$

To help understand the full Pareto frontier of the electric motor design space, all three objectives, mass, efficiency, and torque, were allowed to vary. The final converged three dimensional Pareto frontier, as found with the PMBGA, is displayed in figure 53 The iso-torque lines that represent the ten different electric motors in the family are overlaid on this figure as well. This optimization serves as a baseline for variant performance before introducing family commonality.

Perhaps clearer is the two dimensional Pareto frontier for each of the electric motors, figure 54. Also included in this figure is the final motor mass and efficiency performances from the Dai and Scott baseline study[33].

In addition to understanding the trade-off between the mass and efficiency for the 10 motors, it is

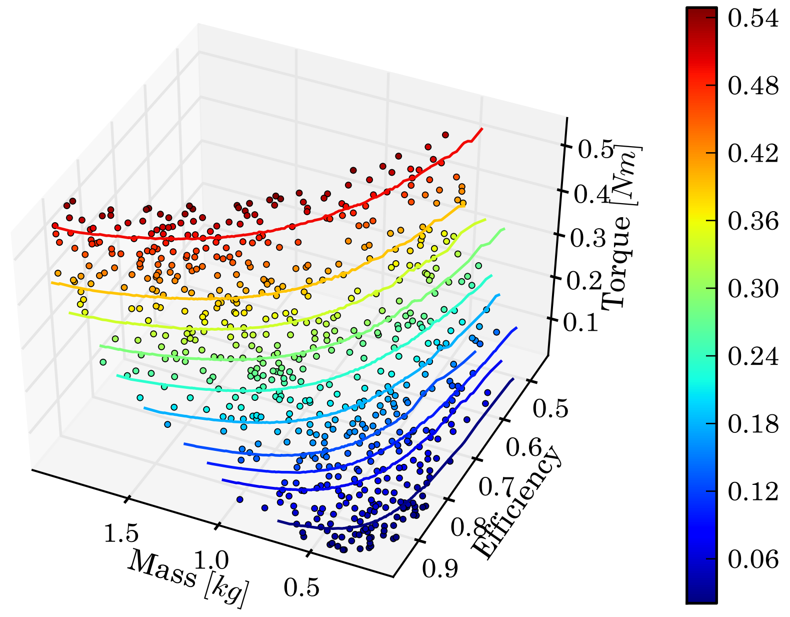


Figure 53: Three Dimensional Pareto Frontier for all Motors

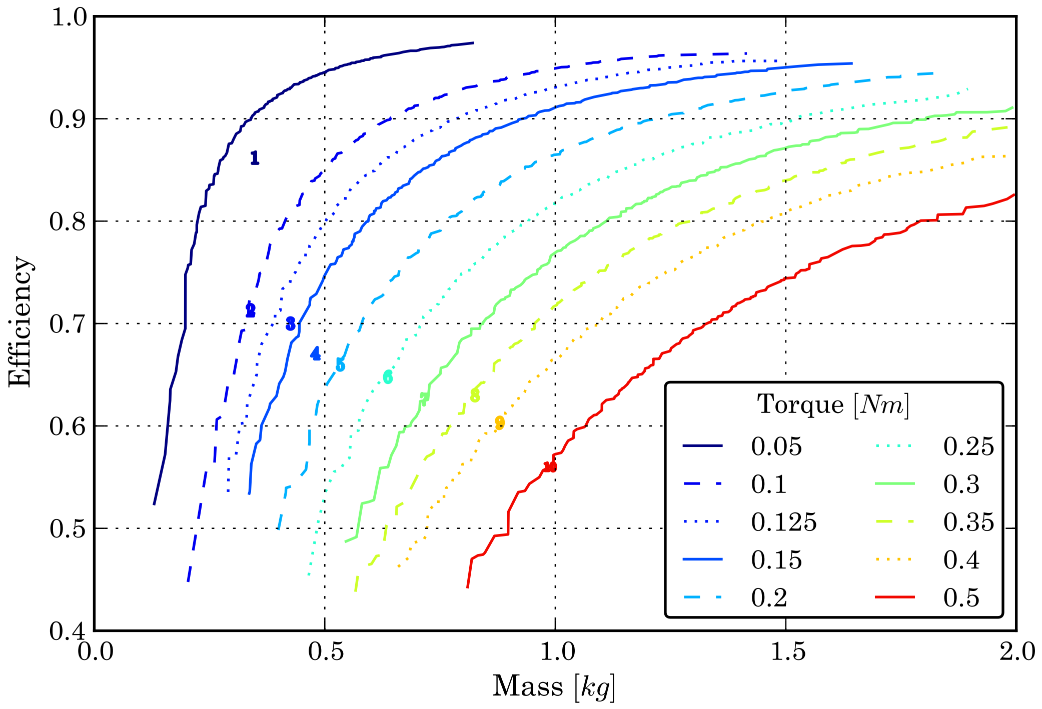


Figure 54: Mass and Efficiency Pareto Frontier for all Motors

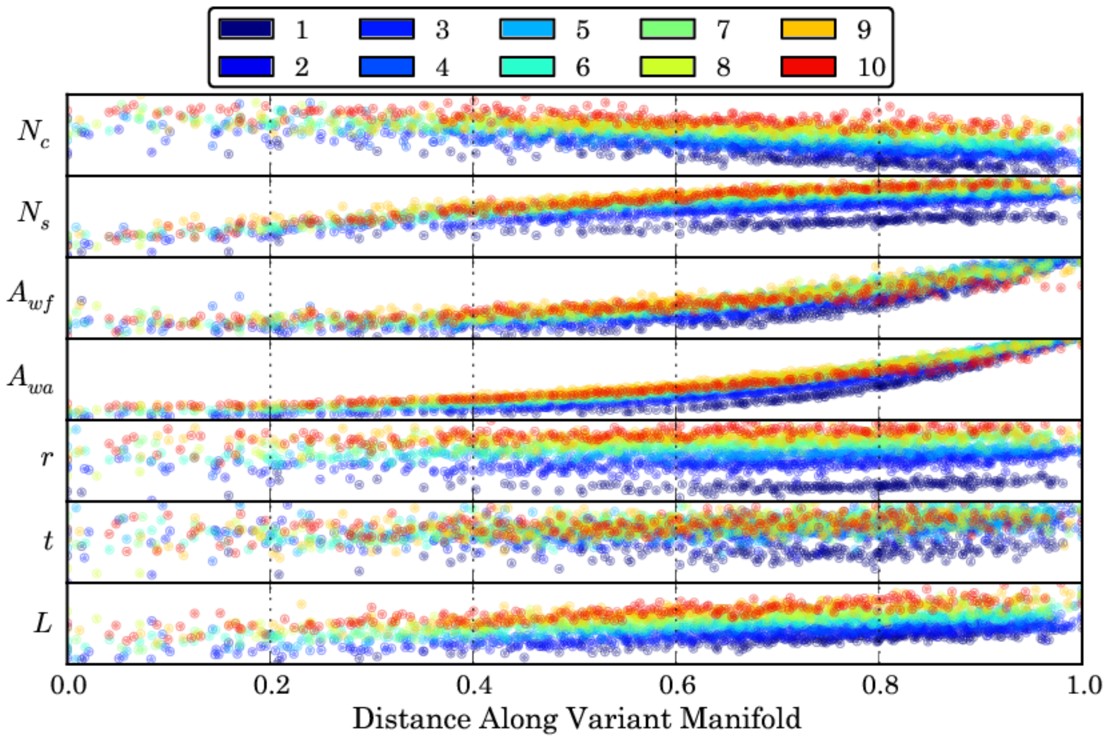


Figure 55: Variation of Motor Variant Design Variables Along Pareto Frontier

important to understand the variation of the design parameters that also occurs along these frontiers, figure 55. For each of the motor Pareto frontiers, the cases were sorted by low efficiency and low mass to high efficiency and high mass. The design variables can then be plotted against their position along their Pareto frontier. This figure demonstrates how smoothly the design variables vary along the frontier. For example, given Pareto optimality and either mass or efficiency all of the other design variables are determined.

In other words, as you move along the frontier, there is a unique mapping back to the combination of Pareto optimal design variables. This helps to demonstrate that there are relatively few degrees of freedom when constrained onto the frontier. Finding significant commonality in the family requires a performance sacrifice from the product's Pareto optimal. Another view of the Pareto frontier is shown in figure 84. This parallel axis plot confirms the strong trend along the Pareto frontier and also reveals some of the intervariable dependance.

The baseline study comes from Dai and Scott[33]. In their study, the motors are optimized individually. Each motor's sensitivity to particular design variables is then determined. This sensitivity

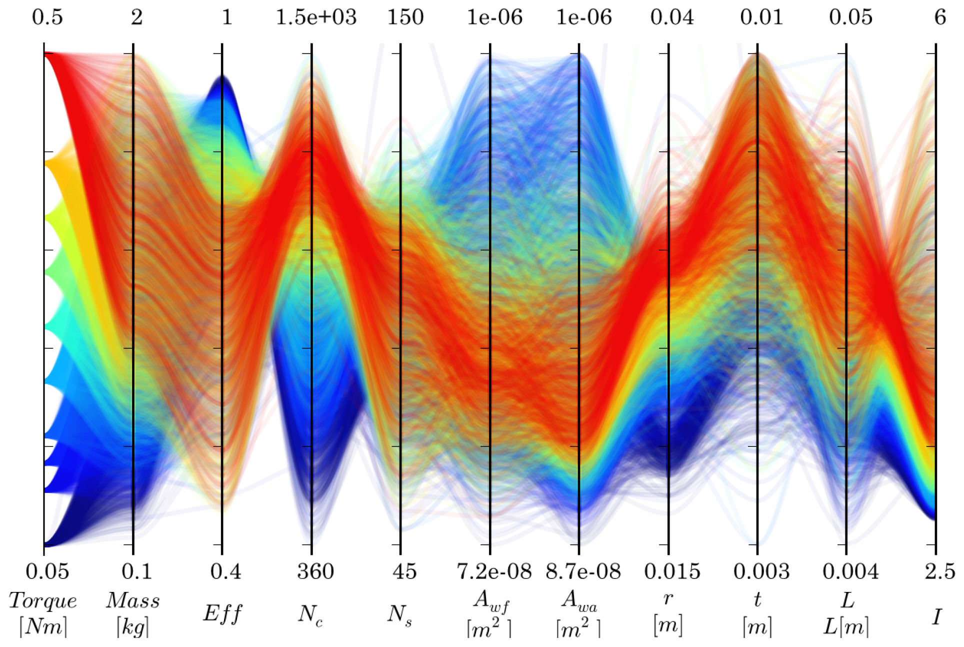


Figure 56: PMBGA Final Front

is clustered and used to construct a dendrogram to help facilitate platform configuration decisions. For benchmarking purposes, the baseline study's platform configuration, design variables settings, and motor performances are shown in table 10.

In the baseline study, Dai and Scott use a slightly different electric motor implementation where the current is not solved for iteratively. Because of this difference in implementation, the design variable settings are evaluated using the Hernandez model and yield slightly different objective values, as seen in table 11. This discrepancy is not an issue, because the commonality approaches

Table 10: Platform and Design Variable Settings from Dai and Scott[33]

Motor	Design Variables							Objectives			
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L	i	η	M	T
1				0.315	15.18			3.03	86.2	.347	.05
2		62		0.19	18.24			3.66	71.3	.338	.10
3				0.208	20.79		20.69	3.73	70.0	.425	.125
4	1,051			0.211	22.44			3.89	67.1	.479	.15
5			0.279	0.222	23.65	6.99		3.95	66	.534	.2
6				0.239	24.07		24.43	4.03	64.8	.637	.25
7		75		0.248	24.88			4.16	62.6	.717	.3
8				0.294	24.76		26.39	4.14	63.0	.826	.35
9	1,246			0.296	25.83			4.33	60.3	.879	.4
10				0.305	27.78			4.66	56.0	.988	.5

Table 11: Implemented Model Platform and Variable Settings

Motor	Design Variables							Objectives			
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L	i	η	M	T
1				0.315	15.18			2.97	87.9	.356	.05
2		62		0.19	18.24			3.57	73.1	.351	.10
3				0.208	20.79		20.69	3.63	71.8	.440	.125
4	1,051			0.211	22.44			3.78	69.0	.496	.15
5			0.279	0.222	23.65	6.99		3.84	67.9	.556	.2
6				0.239	24.07		24.43	3.91	66.6	.663	.25
7		75		0.248	24.88			4.04	64.5	.749	.3
8				0.294	24.76		26.39	4.02	64.8	.861	.35
9	1,246			0.296	25.83			4.19	62.2	.920	.4
10				0.305	27.78			4.50	58.0	1.039	.5

can be benchmarked using the performance objective requirements from the new implementation. If the approaches are able to recover the platform variables estimates near the targets, then it supports the clustering approach's commonality identification usefulness in the generic product family design methodology.

The variables settings from table 11 can be visualized with a parallel coordinate plot discussion of family visualization, figure 57. One interesting trend for motor 1 is that it has a very high value of A_{wa} compared to other low torque motors.

7.3 Testing Commonality Identification Approaches

This dissertation proposes two novel commonality identification approaches for use in the generic product family design methodology. The first is the cluster approach, which uses fuzzy c-means clustering to create component fuzzy similarity relationships for each module space. The second is the probabilistic approach, which uses statistical inference on a Bayesian network surrogate model to extract component similarity relationships. The universal electric motor problem, described above, is a particularly useful tool to test the effectiveness of these novel approaches in identifying component commonality (etc). Specifically, four experiments are performed to allow observations to be made about the two approaches. This section will briefly each experiment.

Sub-hypothesis 2.1

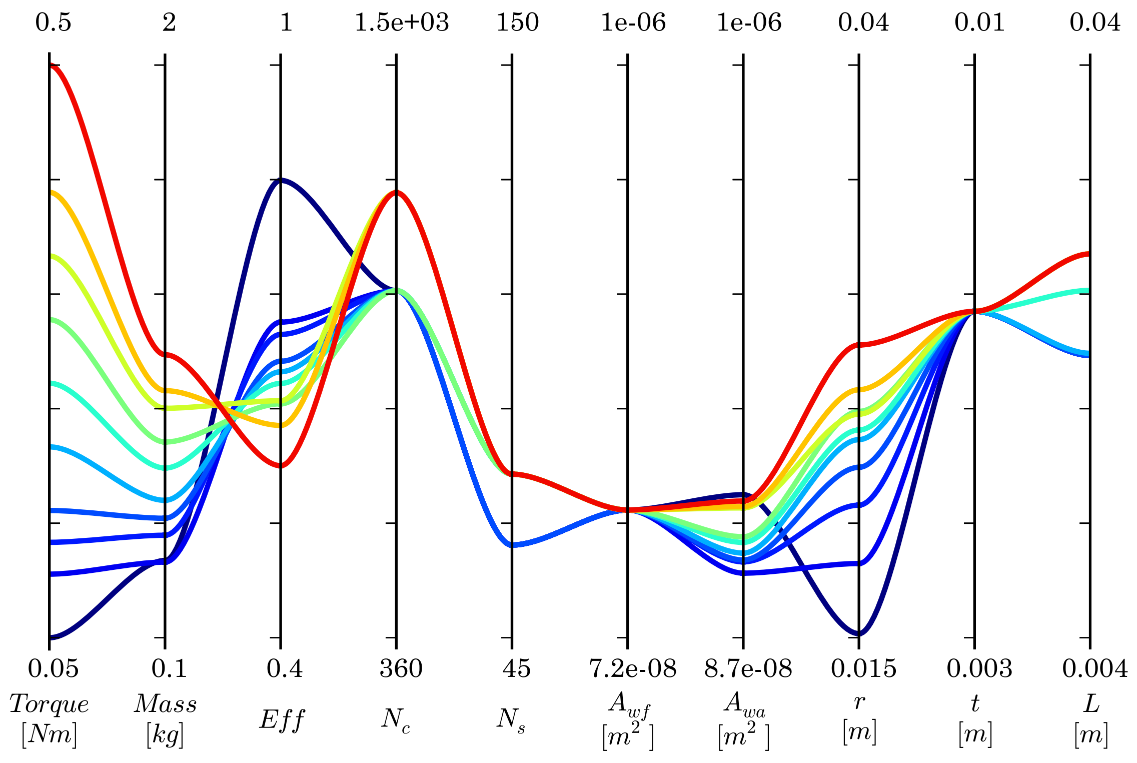


Figure 57: Baseline Family

Sub-Hypothesis 2.1

If a sufficiently accurate and dense database can be generated, then a machine learning pattern recognition technique like fuzzy clustering could be used to help identify component commonality and potentially form a platform.

The general ability of the fuzzy clustering approach to capture commonality is tested in Experiment 1. The experiment uses a baseline product family study of a set of universal electric motors. If the clustering can predict the baseline platform values then it helps to support sub-hypothesis 2.1. One key factor in determining the applicability of the clustering approach is to understand its prediction capability as a function of the database.

Experiment 2 tests the found platform values using three different design space exploration techniques while also varying their sampling density. It is expected that the MC or LHC explorations will perform identically for large numbers of samples as the average density of points will be approximately identical. However for a low number of design points, the LHC will probably have more consistent platform prediction ability while the MC may get “lucky” and have many points near the target product performance ideals. There is another hypothesis relating to the fuzzy clustering approach and it is driven from the observation that product families with long life-cycles need to be flexible to change requirements.

Sub-Hypothesis 3.1

If the design constraints are changed, then using the new feasible subset from the design database and performing the pattern recognition will reveal the sensitivity of component sharing.

Experiment 3 tests this idea of platform sensitivity using a set of different performance targets. If the method can find accurate platform values with different performance targets and partitioning, then it helps to support sub-hypothesis 3.1. Because of the possible scalability issues identified at the end of chapter 4, another commonality identification approach is formulated, sub-hypothesis 2.2.

Sub-Hypothesis 2.2

If a model can be generated that encodes the joint probability distribution, then component similarities can be inferred given performance constraints.

Experiment 4 again uses the same product family design study used in experiment 1 and builds a Bayesian network representation of the Pareto frontier. This Bayesian network can use the design studies performance targets and platform configuration as a starting point for the commonality reasoning. If the probabilistic approach yields design variable values with expected values near the baseline study, then it supports sub-hypothesis 2.2. Similarly with the clustering approach, the probabilistic approach needs to be able to identify flexible commonalities for the family, sub-hypothesis 3.2.

Sub-Hypothesis 3.2

If the design constraints are changed, then any changes to the posterior distributions from the probabilistic model will reveal the sensitivity of component sharing

Hypothesis 3.2 is supported though the Bayesian networks ability to accurately capture the design space joint distribution and further tested in the aircraft family demonstration problem.

7.3.1 Experiment 1

Experiment 1 benchmarks the clustering approach to the above referenced baseline study. The first part is a qualitative comparison of the equivalence hierarchy to the baseline dendrograms. The second part uses the baseline platform configuration to make a quantitative comparison between the estimated design variable settings and the baseline targets. This comparison uses a database generated by a uniform Monte Carlo (MC) design space exploration. The MC exploration generated 10,000 cases for each motor for a total database size of 1,000,000 design cases.

The database is then filtered using the motor target η and M objective values from table 11. Filtering the database yields a feasible subset of only the best design alternatives based on the highest fitness to the target objective values. This prunes the inferior generated designs so they do not affect the clustering results. The fitness for point i, equ 78, is determined by its Euclidean distance to the target values θ for all of the dimensions d.

Using the best design alternatives subset, each design's weight, w_i, is calculated by normalizing the fitness and then dividing by the sum, equ 80:

$$fitness\quad_i = \sqrt{\sum_j^d \left(\hat{\theta}_j - \theta_j\right)} \tag{78}$$

$$w_i^* = \frac{fitness - \min\left(fitness\right)}{\max\left(fitness\right) - \min\left(fitness\right)} \tag{79}$$

$$w_i = \frac{w_i^*}{\sum_i^n w_i^*} \tag{80}$$

where n is the number of design alternatives per product remaining in the database.

Next, each module space, $\mathbb{S} = \left\{\{N_c\}, \{N_s\}, \{A_{wa}\}, \{A_{wf}\}, \{r\}, \{t\}, \{L\}\right\}$, is extracted from the database. While a module can, in general, be described by any number of design parameters, depending on the functional breakdown and product architecture, each design parameter in this problem is its own module.

Because the number of clusters in each module subspace is not known, each subspace undergoes a series of fuzzy c-means clusterings using a varying number of clusters. Figure 58 shows the Xie

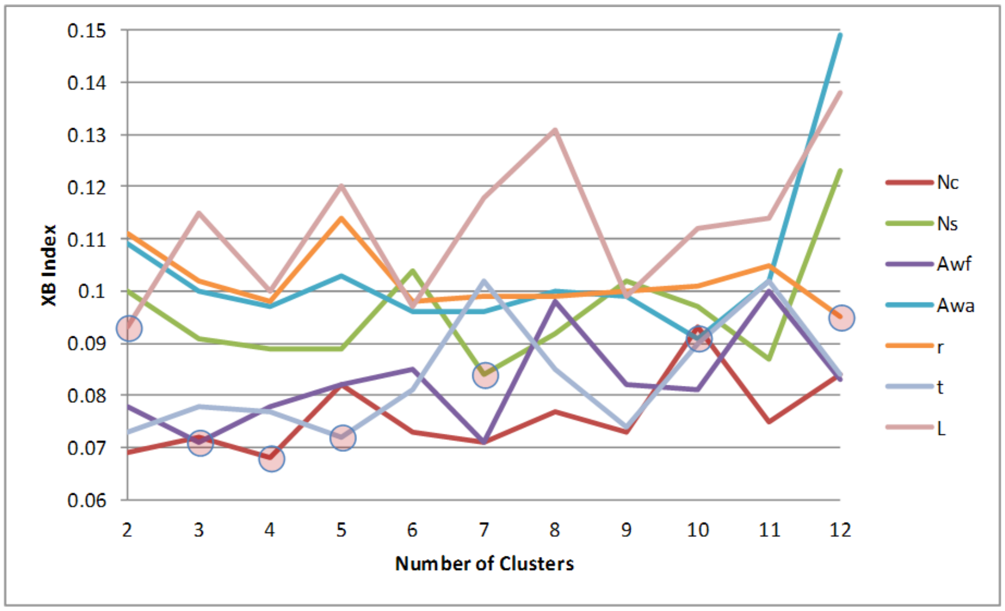


Figure 58: XB Index of Each Module for Different Number of Clusters

Beni (XB) cluster validity index of all of the different number of clusters for each module. The circled points show the lowest XB index for each module and corresponds to the optimal number of clusters to use to extract product fuzzy similarity relations.

Using the membership functions with the optimal number of clusters for each module, the weighted average product-to-cluster membership functions are calculated, equ 9. Fitness based weighting ensures that better performing design alternatives contribute more of their membership function to the average. This step is needed to collapse all of the different design alternatives for a given product into a single membership function.

For each module, binary product fuzzy relationships are calculated using equ 10. The binary fuzzy relationship is highest between those products with a high membership in the same clusters. As an example, table 12 shows the binary fuzzy relationship of *L*.

To be able to extract equivalence classes, the binary fuzzy relationship matrices are made transitive. Using the transitive closures, equ 15, an example of the binary equivalence relationship for module *L* can be seen as an example in table 13.

It is difficult to understand the equivalence relationships matrix directly. However, the equivalence hierarchies can be shown as dendrograms where α corresponds to the degree of similarity in the fuzzy equivalence relationship matrix, figures 59-65. Again a family platform configuration

Table 12: *L* Binary Fuzzy Similarity Relation

	Product									
Product	1	2	3	4	5	6	7	8	9	10
1	1.000	0.967	0.916	0.899	0.810	0.764	0.584	0.538	0.512	0.420
2	0.967	1.000	0.949	0.932	0.843	0.797	0.618	0.572	0.545	0.453
3	0.916	0.949	1.000	0.983	0.894	0.848	0.669	0.623	0.596	0.504
4	0.899	0.932	0.983	1.000	0.911	0.865	0.686	0.640	0.613	0.521
5	0.810	0.843	0.894	0.911	1.000	0.954	0.774	0.728	0.702	0.610
6	0.764	0.797	0.848	0.865	0.954	1.000	0.820	0.774	0.748	0.656
7	0.584	0.618	0.669	0.686	0.774	0.820	1.000	0.954	0.928	0.836
8	0.538	0.572	0.623	0.640	0.728	0.774	0.954	1.000	0.974	0.882
9	0.512	0.545	0.596	0.613	0.702	0.748	0.928	0.974	1.000	0.908
10	0.420	0.453	0.504	0.521	0.610	0.656	0.836	0.882	0.908	1.000

Table 13: *L* Fuzzy Equivalence Matrix

	Product									
Product	1	2	3	4	5	6	7	8	9	10
1	1.000	0.967	0.949	0.949	0.911	0.911	0.820	0.820	0.820	0.820
2	0.967	1.000	0.949	0.949	0.911	0.911	0.820	0.820	0.820	0.820
3	0.949	0.949	1.000	0.983	0.911	0.911	0.820	0.820	0.820	0.820
4	0.949	0.949	0.983	1.000	0.911	0.911	0.820	0.820	0.820	0.820
5	0.911	0.911	0.911	0.911	1.000	0.954	0.820	0.820	0.820	0.820
6	0.911	0.911	0.911	0.911	0.954	1.000	0.820	0.820	0.820	0.820
7	0.820	0.820	0.820	0.820	0.820	0.820	1.000	0.954	0.954	0.908
8	0.820	0.820	0.820	0.820	0.820	0.820	0.954	1.000	0.974	0.908
9	0.820	0.820	0.820	0.820	0.820	0.820	0.954	0.974	1.000	0.908
10	0.820	0.820	0.820	0.820	0.820	0.820	0.908	0.908	0.908	1.000

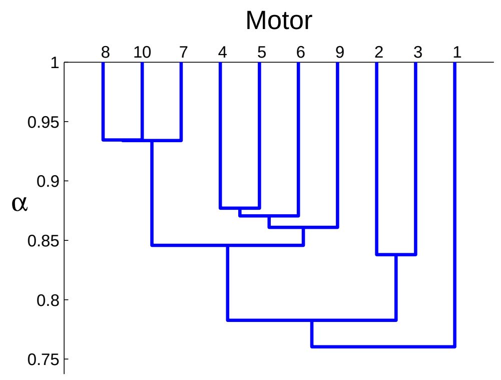


Figure 59: A_{wa} Partitions

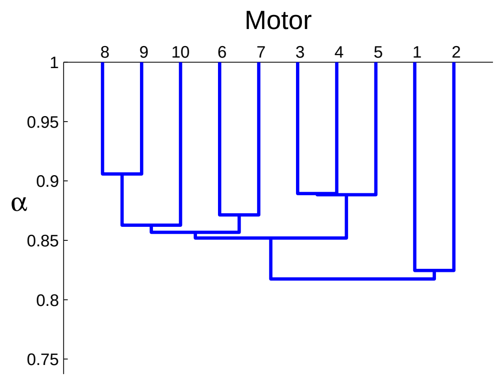


Figure 60: r Partitions

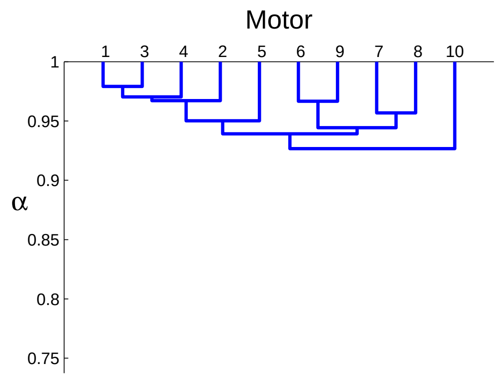


Figure 61: A_{wf} Partitions

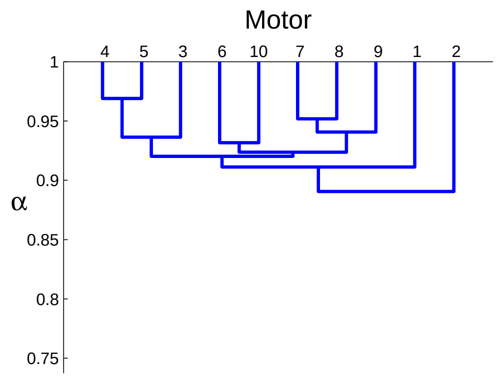


Figure 62: t Partitions

can be generated by taking the appropriate α-cuts on the fuzzy equivalence relations. In analyzing the different equivalence hierarchies, it is expected that products are most similar to products that have similar torque requirements. For example, low number motors have low torque requirements and we would expect to see them more closely linked to each other. Additionally, the α-cuts shows information about how tightly coupled different components are together. Groups that remain unchanged over large ranges of α are stronger than groups that change rapidly over small ranges of α.

The partition hierarchy dendrograms can be compared to the hierarchy clustering from the sensitivity analysis work done by Dai and Scott[33]. The motor family was very sensitive to A_{wa} and r causing them to be unique for each motor. A_{wf} and t were the most insensitive modules they

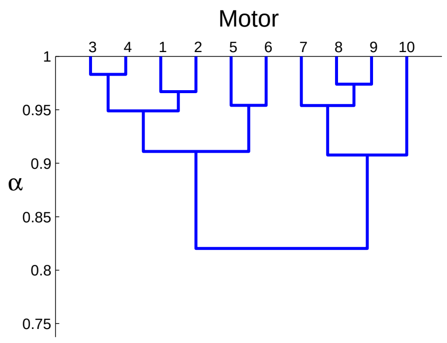


Figure 63: *L* Partitions

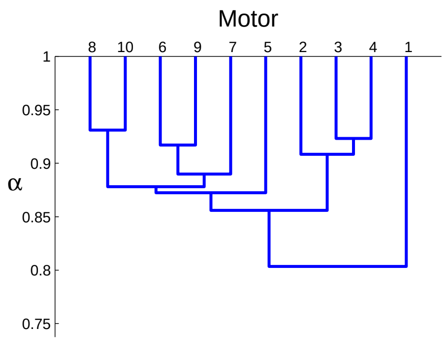


Figure 64: N_s Partitions

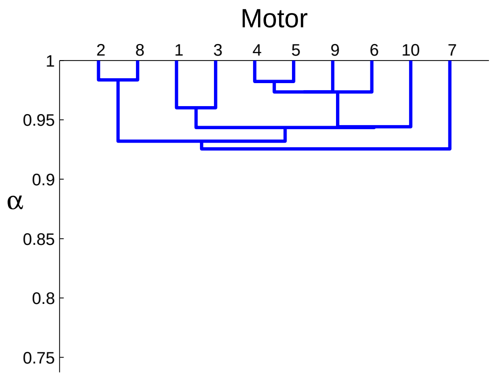


Figure 65: N_c Partitions

found, so it was decided to share them across the whole family. N_s,N_s, and L are in between the two extremes and are platforms shared across subsets of the family.

Likewise, the clustering approach formulated in this dissertation also identifies that the products are highly sensitive to armature wire cross section area A_{wa} and motor radius r; figure 58 shows a high number of clusters needed, and the partition hierarchies in figures 59 and 60 require a relatively low threshold (.92) before products start to get grouped.

The field wire cross section area A_{wf} and the stator thickness t, shown in figures 61 and 62 respectively, seem to be insensitive to the different products as they are pretty much fully grouped around $\alpha = 0.9$. Figure 63 showing the stack length L also closely corresponds to the dendrograms presented in the baseline study. The largest difference is in motor six which seems to be strongly grouped with motors one through five rather than independent. The number field turns N_s, figure 64, shows three strong partitions, {{1},{2,3,4},{5,6,7,8,9,10}} which is similar to the sharing used in the baseline study. The number of armature wire turns N_c, figure 65, appears to have a very low total grouping threshold like A_{wf} and t which does not conform as well as the other modules' results. This could be due to the database not being densely sampled enough around the target motor settings.

Overall, results seem similar and behave as expected with the dendrograms presented by the baseline study. These sharing hierarchies offer insight into product component configurations, and can be used to aid decision makers in further exploration of the family. For example, studying product families that share stack length, radius, or field windings across the whole family will probably lead to the worst individual product performances. Sharing the other design variables, on the other hand, will probably cause the least performance trade-offs.

Next, using the specified baseline platform configuration, the clustering approach steps will be shown in sequence, using the database created from a uniform Monte Carlo sampling over the entire design ranges noted in table 9. For each of the ten motors 100,000 cases are generated for a total database size of 1,000,000 points. The database is filtered using objective target performance requirements from table 11.

The design cases are then filtered using a Euclidean distance fitness to select the closest points to the target motor performances. Figure 66 shows this feasible subset from the database on a parallel coordinate plot. Additionally, the more opaque the point, the closer it is to its target. The use of the parallel coordinate plot allows visual feedback into the bandwidths of particular design variables for each motor. For instance, the plot confirms the motors are rather insensitive to the variables A_{wf} and t. This agrees with Dai and Scott, who use those variables as a platform across the whole family. As part of the fuzzy clustering approach, figure 28, the weighted and filtered data is clustered and processed to yield similarity relations. To the right of the parallel plot is a weighted graph showing these fuzzy similarity relationships. The darker the line the stronger the relationship is between those products in that module space. The weighted graphs again confirm that the products are rather similar across the module space because there are more connections across the products.

Because of the insensitivity of the variables A_{wf} and t to the motor performance, they are made platform variables. The database is refiltered and weighted to select the points closest to the performance requirements and the platform average values of A_{wf} and t. This new weighted subset is again clustered and processed to try and identify the impact of making A_{wf} and t common, figure 67. This new parallel plot shows a slightly different subset and has decreased the bandwidth for both A_{wf} and t. In doing so, the clustering reveals some slightly different similarity relationships in the weighted graph. There appears to be a strong partitioning for the stack length according to the baseline study $\{\{1,2,3,4,5\},\{6\},\{7,8,9,10\}\}$.

By making L a platform variable the process is again repeated, figure 68. The new database subset does appear to change the similarity relationship. However, the weighted graph alone does not indicate obvious, crisp partitionings. In general, though, trends in the weighted graph do seem to follow the expected behavior. Motors are differentiated by their torque requirement and motors with more similar torque appear to have design parameters that are more similar than those with more dissimilar torque.

To complete the final baseline platform configuration, the partitionings for the N_c, and N_s variables are included, figure 69. Again, the structures appear to have the appropriate trends, but the bandwidth of the design variables remains large.

To compare baseline target values, a single value that indicates the relative closeness of the feasible subset is used. To calculate this estimate, the Relative Root Mean Squared Deviation (RRMSD) is used, equ 81:

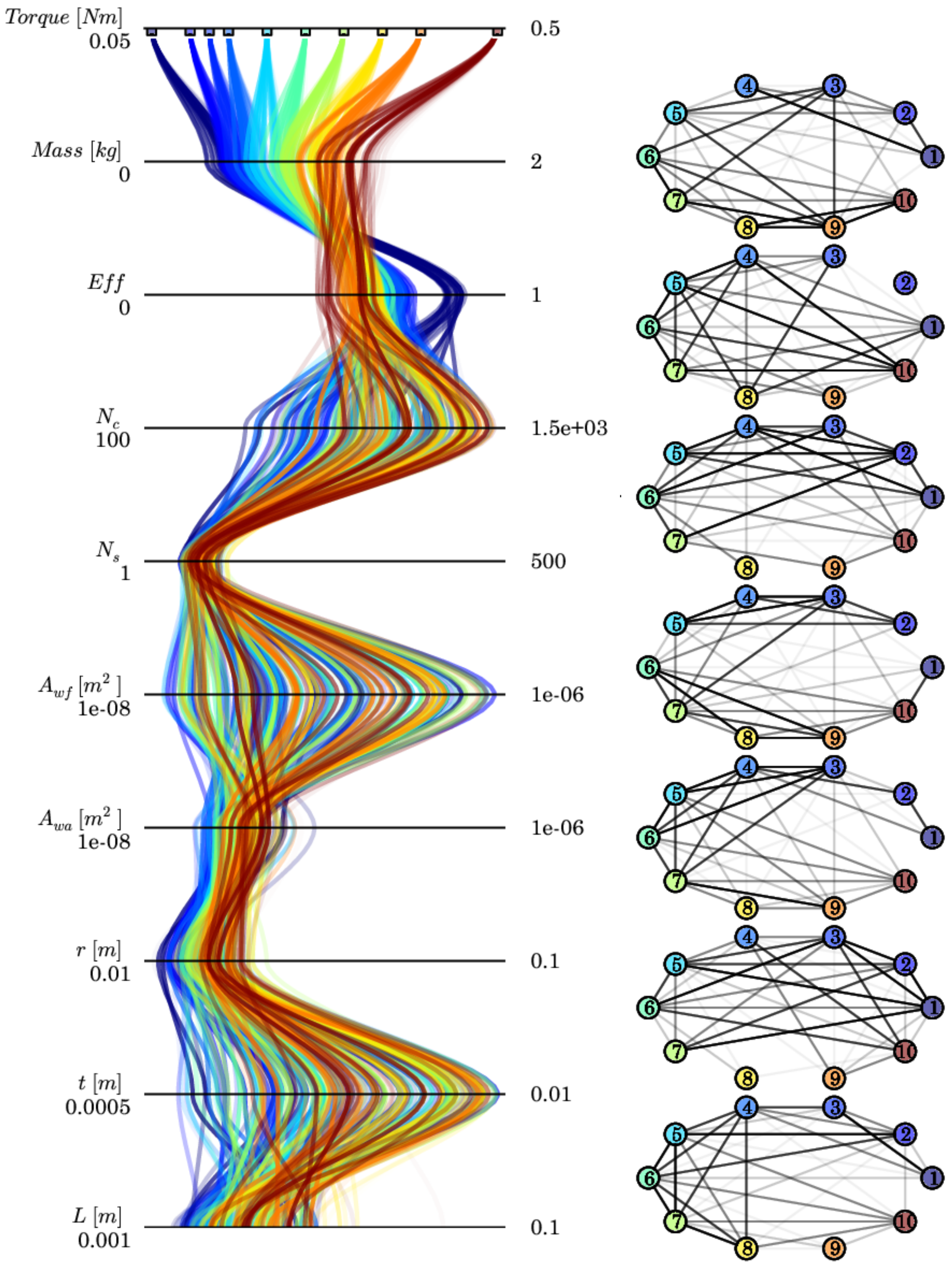


Figure 66: 1,000,000 Filtered MC

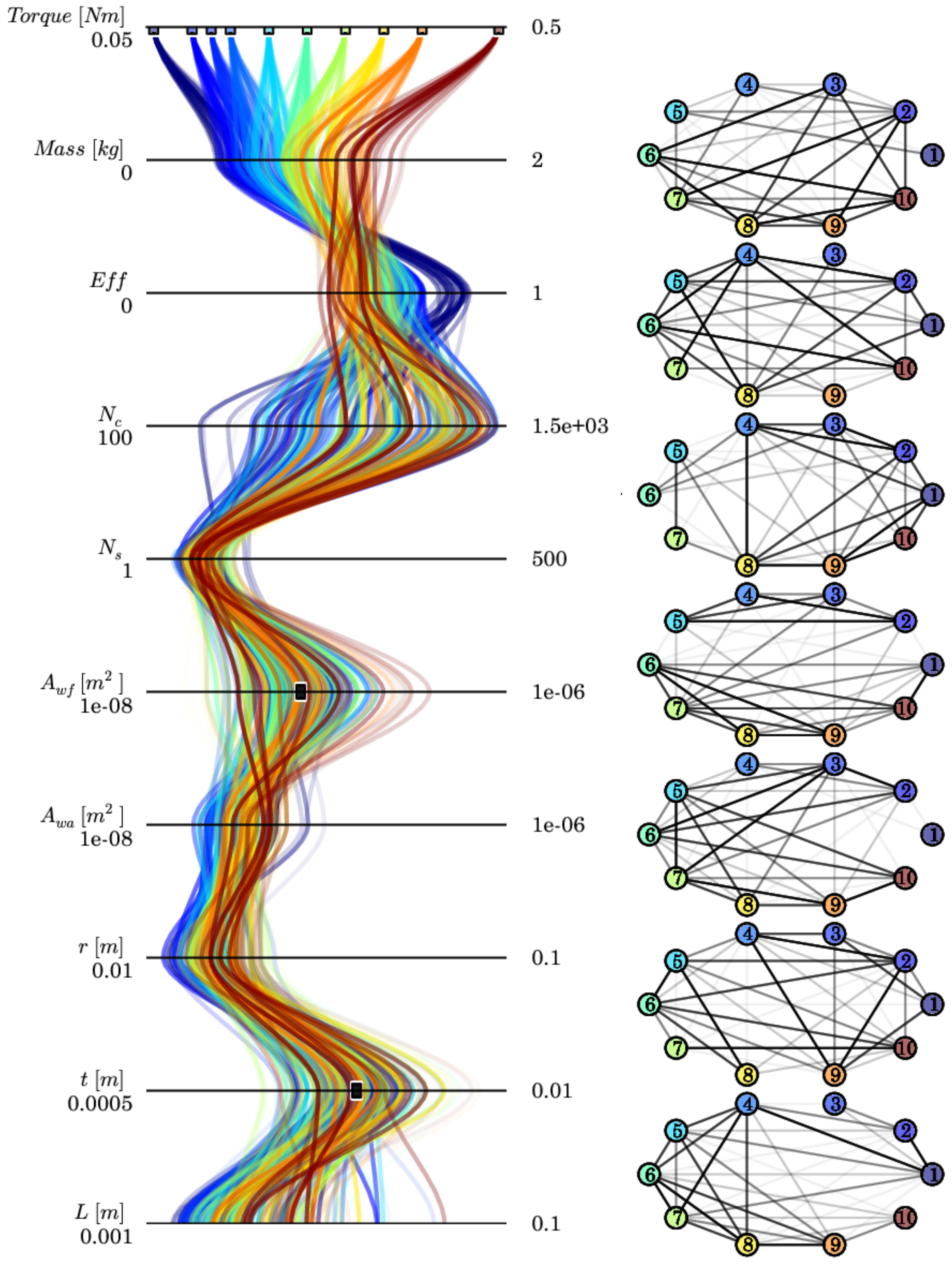


Figure 67: 1,000,000 Filtered MC with A_{wf} and t Common

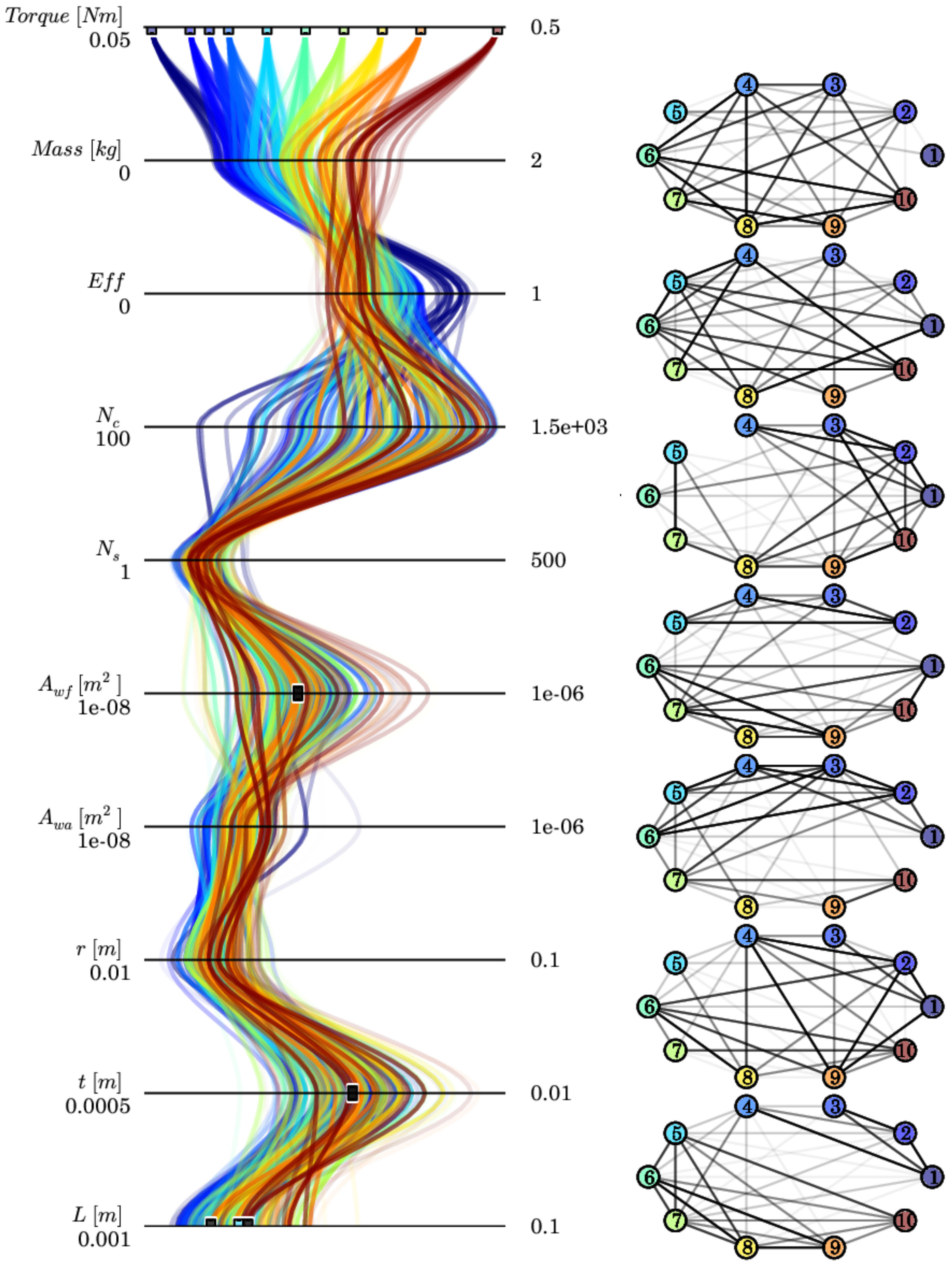


Figure 68: 1,000,000 Filtered MC A_{wf} , t, and L platforms

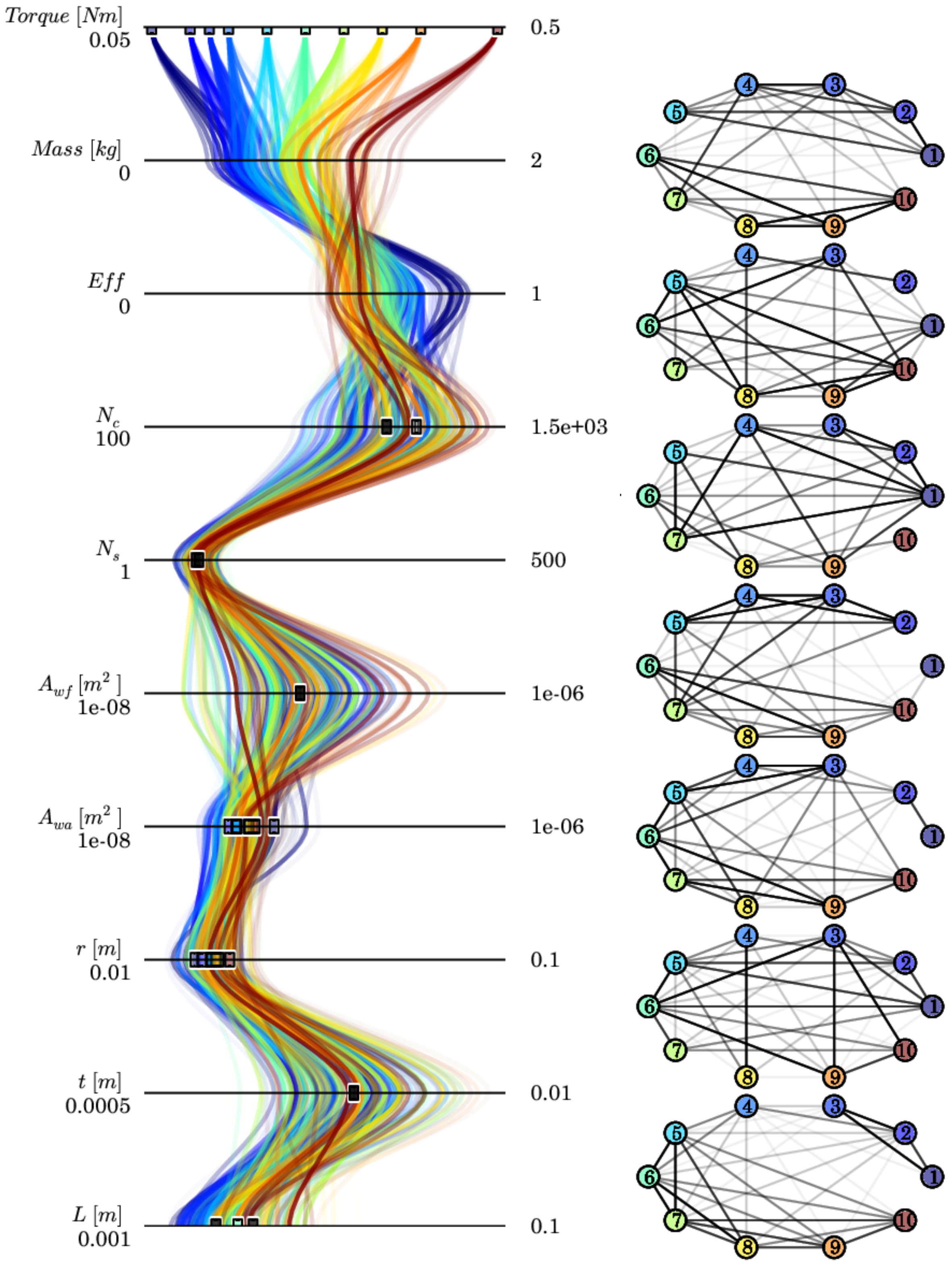


Figure 69: 1,000,000 Filtered MC A_{wf} , t, L ,N_c, and N_s platforms

Table 14: RRMSD of the MC Design Space Exploration

Motor	Design Variables						
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L
1	0.160	0.235	0.492	0.187	0.312	0.298	0.442
2	0.160	0.235	0.492	0.220	0.275	0.298	0.442
3	0.160	0.235	0.492	0.184	0.259	0.298	0.442
4	0.160	0.235	0.492	0.172	0.189	0.298	0.442
5	0.160	0.203	0.492	0.147	0.159	0.298	0.442
6	0.160	0.203	0.492	0.200	0.116	0.298	0.276
7	0.160	0.203	0.492	0.147	0.154	0.298	0.310
8	0.147	0.203	0.492	0.101	0.203	0.298	0.310
9	0.147	0.203	0.492	0.137	0.131	0.298	0.310
10	0.147	0.203	0.492	0.112	0.120	0.298	0.310

$$RRMSD\left(\hat{\theta}\right) = \frac{\sqrt{\frac{\sum_{i=1}^{n} w_i\left(\hat{\theta}_i - \theta\right)^2}{\sum_{i=1}^{n} w_i}}}{\theta} \tag{81}$$

The RRMSD compares the design variable value $\hat{\theta}$ to the baseline design variable target θ for each point i in the weighted feasible subset of the database. Lower values indicate that the "bundle" of design alternatives is near the target with a small spread. The RRMSD of the feasible subset of the 1,000,000 MC database to the baseline target values is shown in table 14. More detail of this estimate can be found in figure 74. These histograms show the distribution of the feasible subset for the different platforms. The platform distributions are shown to be centered around the baseline target values, θ.

It is hypothesized that the clustering approach requires a dense and high quality design database. Figure 79 shows the comparison of the feasible subset objective space, using only performances (left) and then effect of including the average of the design variables for the platform estimate (right). This figure helps to demonstrate the sparsity of the design space exploration. On the left, the points that are included in the subset of points to be clustered are not all that tightly grouped around their performance target. When subsequent variables are partitioned and included in the criteria for subset selection, the sparsity problem becomes much worse as shown on the right.

Because the design database is sparse, another uniform Monte Carlo exploration is performed using refined design variable settings obtained using the feasible subset ranges of the original filtered MC exploration. The new database uses an additional 100,000 cases per motor to yield a denser

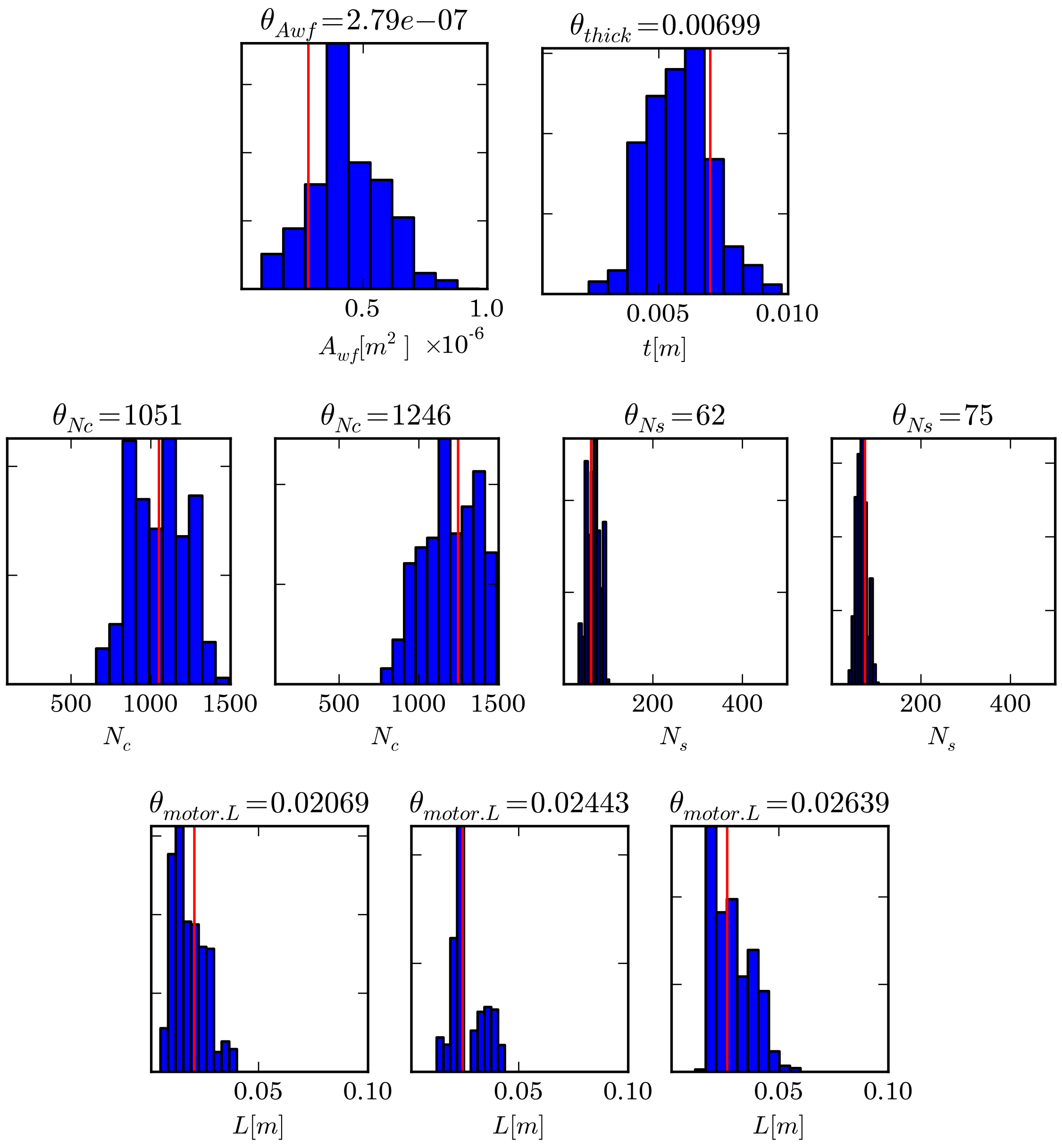


Figure 70: Feasible Subset Platform Histograms for MC Exploration

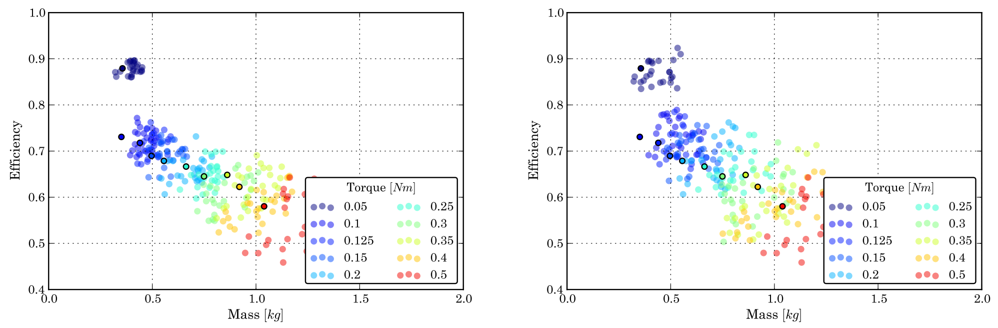


Figure 71: Comparison of Objective. Only performance (left), with design variable partitions (right)

Table 15: Platform and Design Variable Settings

Motor	Design Variables						
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L
1	986	69	0.382	0.308	16.68	5.99	18.44
2	986	69	0.382	0.188	20.49	5.99	18.44
3	986	69	0.382	0.203	21.97	5.99	18.44
4	986	69	0.382	0.201	24.13	5.99	18.44
5	986	75	0.382	0.212	24.50	5.99	18.44
6	986	75	0.382	0.232	25.32	5.99	23.94
7	986	75	0.382	0.244	24.28	5.99	28.18
8	1,265	75	0.382	0.292	24.55	5.99	28.18
9	1,265	75	0.382	0.295	25.51	5.99	28.18
10	1,265	75	0.382	0.301	27.48	5.99	28.18

sampling of points around the target performance values.

Applying the same partitioning for the design variables as the baseline study again yields a much better understanding of the partition values, figure 72. The banded overlapping of the product components are clearly visible because the design variable bandwidths are reduced.

The tightness of the points in the objective space shows how much denser the refined bounds yielded, figure 73. The left half shows the objective space with many more sample points near the target values; the right half shows the final subset by including the platform variables in the selection. While the data could be more densely sampled the spread of the data has been significantly improved with the refined bounds.

The estimate for the platform variables using the clustering approach for the refined MC exploration are shown in table 15.

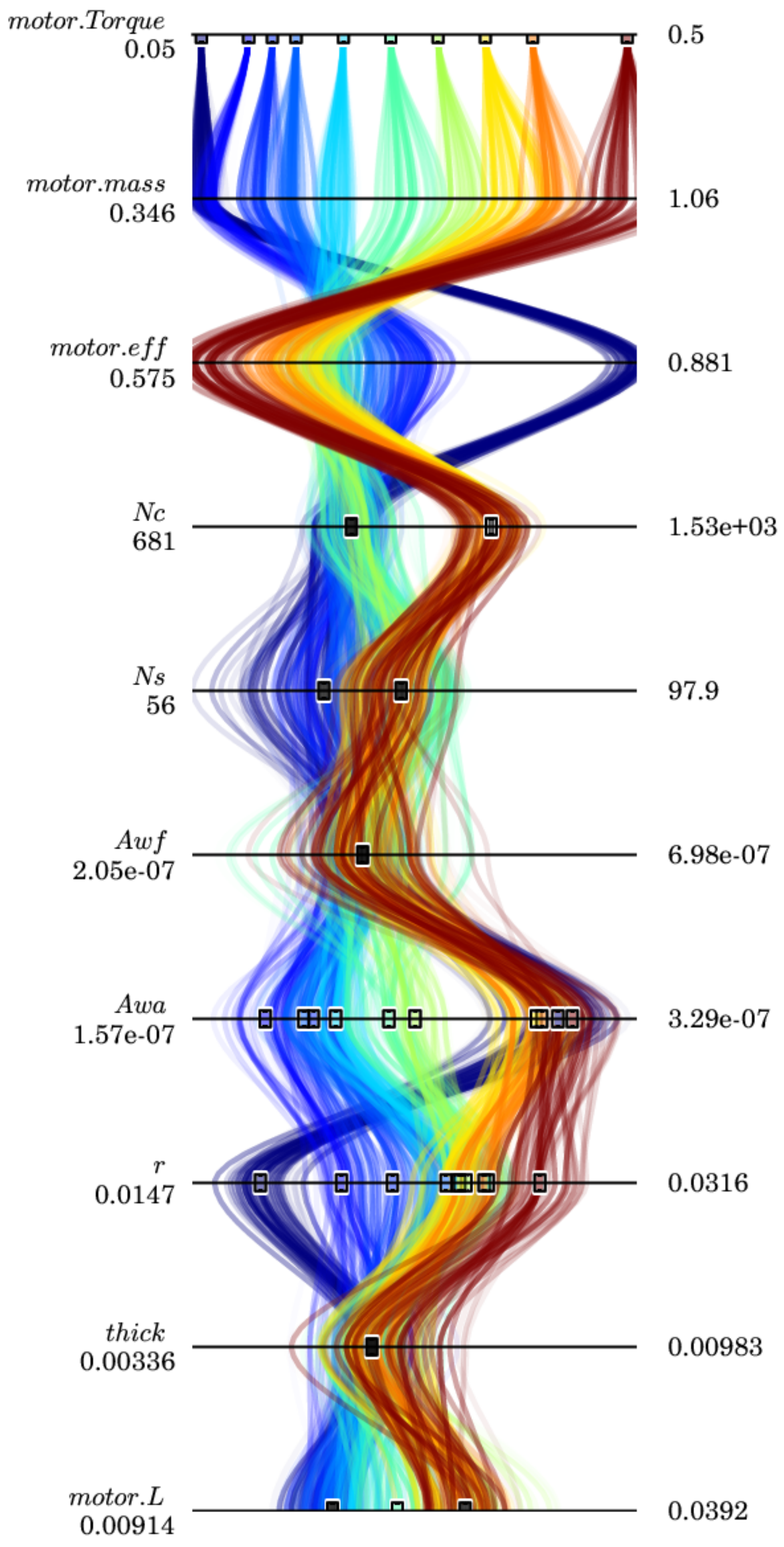


Figure 72: Refined MC 1,000,000 Filtered MC

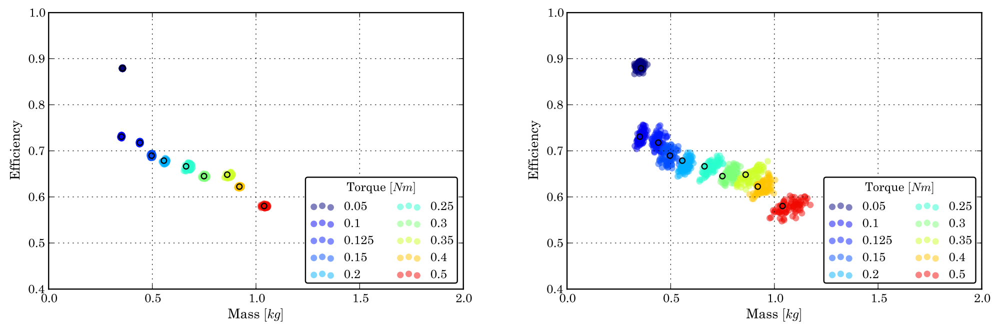


Figure 73: Comparison of Objective

Table 16: Expanded MC RRMSD

Motor	Design Variables						
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L
1	0.072	0.122	0.393	0.039	0.105	0.154	0.168
2				0.030	0.125		
3				0.038	0.066		
4				0.056	0.084		
5		0.040		0.052	0.044		
6				0.044	0.056		0.096
7				0.026	0.040		0.077
8	0.032			0.030	0.027		
9				0.025	0.027		
10				0.025	0.028		

Table 16 shows the design variables' RRMSD of the expanded MC feasible subset to the baseline targets. Overall the lower RRMSD values indicate the feasible subset is closer than the original MC. However A_{wf} remains high and is probably caused by it being a weak influence on the overall motor performance. The platform histograms using the feasible subset are shown in figure 74. Similar to the previous platform histograms, these indicate a much tighter grouping of points in the subset with the distributions near the baseline targets, θ. Notably, the histogram for A_{wf} and t appears shifted from their baseline targets.

The results of this experiment indicate that the fuzzy clustering approach can capture commonality between the different components, and that increasing the design space exploration sampling density can make these relationships clearer. Experiment 2 expounds upon the clustering approach's dependency on the quality and density of database.

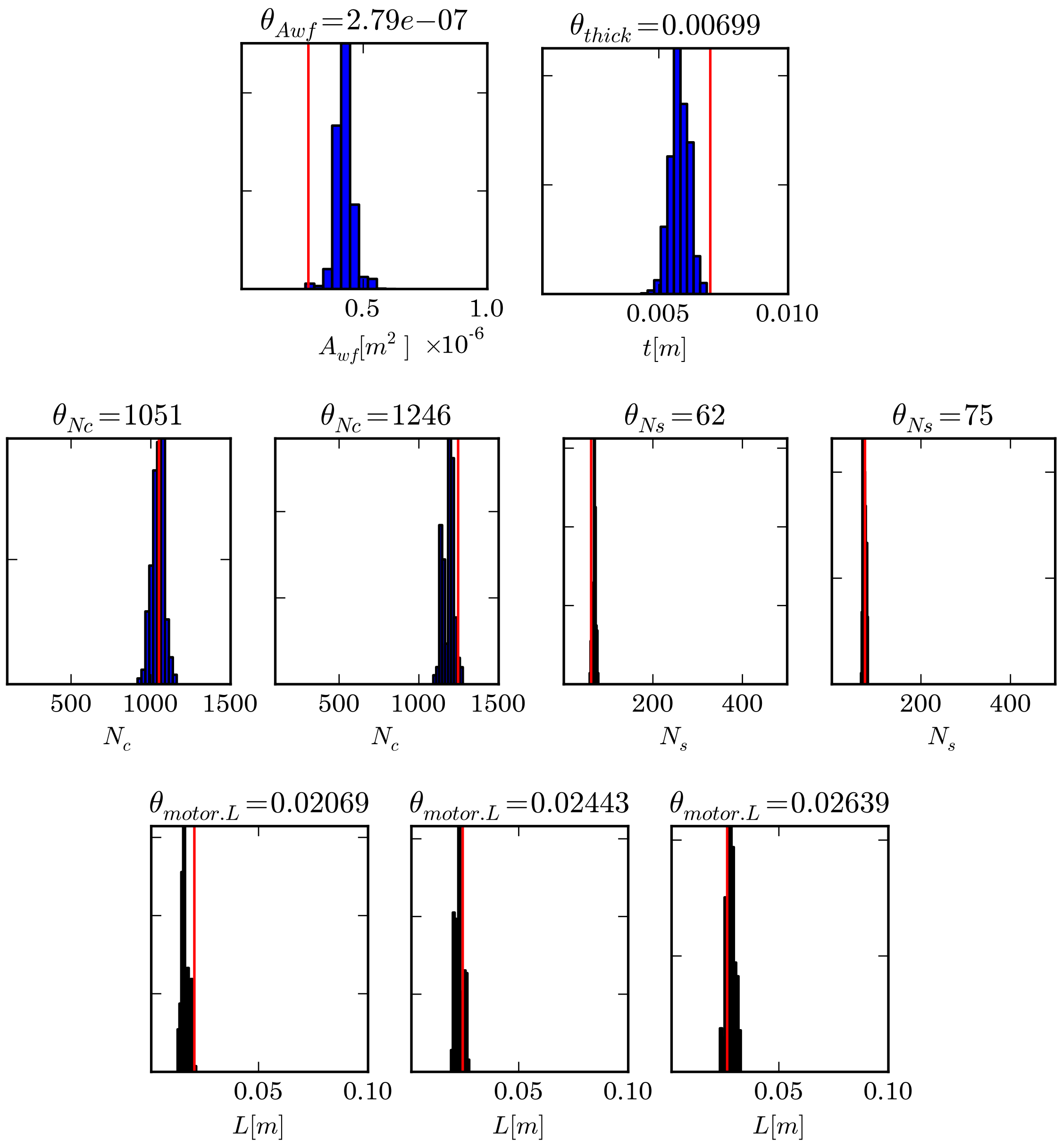


Figure 74: Feasible Subset Platform Histograms for Expanded MC Exploration

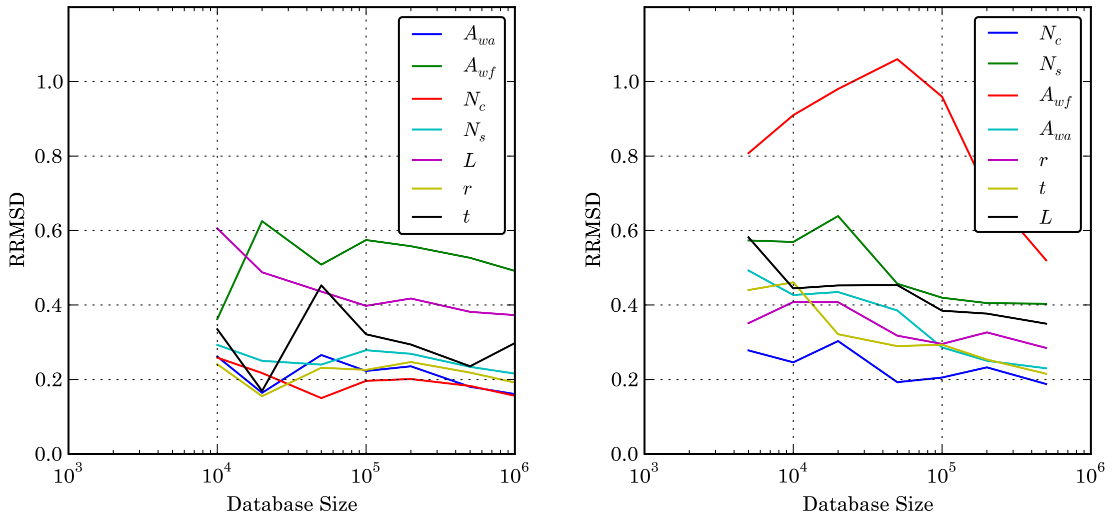


Figure 75: MC left, LHC right

7.3.2 Experiment 2

From experiment 1, the clustering approach seems to be sensitive to the size of the database. This experiment compares three design space exploration methods: Latin Hypercube (LHC), Monte Carlo, and a multiobjective genetic algorithm (MOGA). Using a structured space filling DoE like the LHC, may yield better coverage of the design space when compared to the MC exploration and have more consistent platform estimates. Alternatively, the MOGA should increase the density of points in the database near the Pareto frontier. Figure 54 confirms that the motors' performances in the baseline family remains near their Pareto optimal.

Using the platform estimating process as performed in Experiment 1, figure 75 shows a side by side comparison of each modules average RRMSD for the LHC and MC as the database increases in size. One interesting effect with the MC is a large improvement in the RRMSD at around 20,000 cases. This dip is caused by the MC cases getting "lucky" with a dense sampling of points near the target performances. The LHC trend is more consistent, with a general improvement in error as the number of cases increase. However, once the database gets up to 1,000,000 points both explorations have around the same RRMSD.

It was learned in Experiment 1 that the design variable ranges may need to be refined, figure 76. The left figure shows another MC exploration using the refined bounds. Here the LHC has

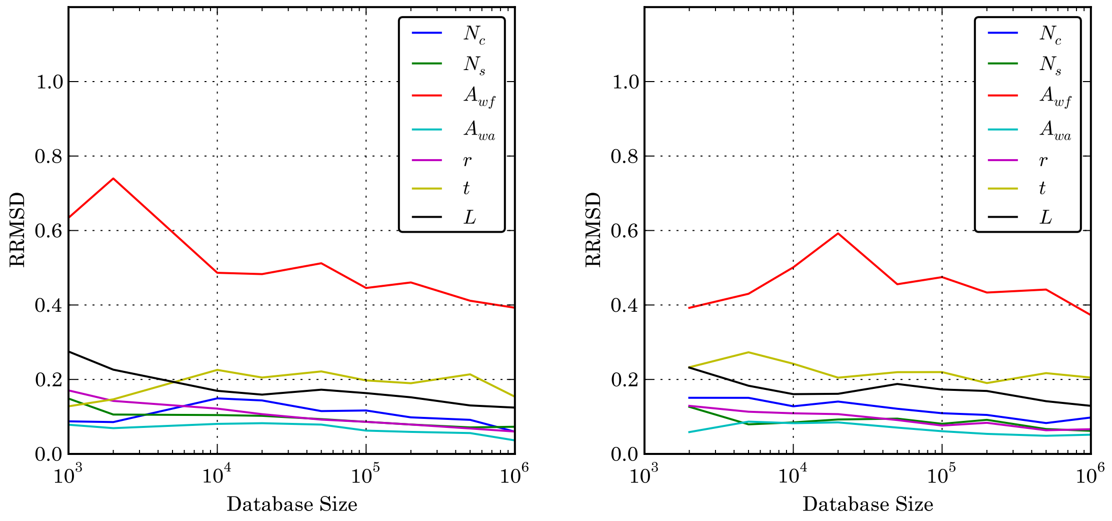


Figure 76: Expanded MC left, LHC right

improved RRMSD and a more consistent improvement compared to the expanded MC exploration. When the database size reaches 1,000,000 points both methods yield similar errors.

From these explorations, it is learned that the unguided exploration using the full design variable ranges leads to a wide platform variable estimate using the clustering approach. The estimated distributions are much tighter when the database density increases.

Unlike the unguided LHC and MC explorations, the MOGA PMBGA builds on knowledge of previous good solutions to determine where to explore next. Figure 77, shows the feasible subset RRMSD repeated using the iteration history of the PMBGA. The RRMSD tends to jump with relatively few generations being evaluated. However, as the population continues to evolve the front is converged and the RRMSD settles. After the database grows to around 100,000 cases, there is an indication that the RRMSD for N_S stop decreasing and start increasing. This is because the population continues to improve the Pareto optimal and move past the design variable region of the commonality needed for the family. In other words, the family members are “near” the frontier while not being exactly on it.

In all of the explorations, the RRMSD of A_{wf} is significantly higher than the other design variables especially for the the unguided explorations. However, using the PMBGA, the RRMSD for A_{wf} falls to around 0.3 and is more consistent with the other modules.

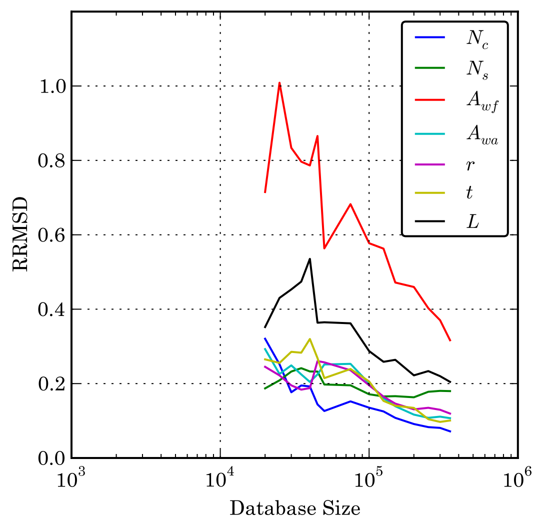


Figure 77: Design Variable Prediction Errors using PMBGA History

The subset for the final generation is shown in the parallel plot, figure 78. The parallel plot also shows the same general trend as the baseline family parallel plot, figure 57. However, it is not as clearly shown as in the parallel plot for the refined MC exploration, figure 72

Figure 79 shows the change in the spread of the feasible subset used for clustering. On the left is the subset using only the performance targets to select the points. The right shows the subset of the data with all of the baseline partition configurations active. Now if the target performance points were farther from the Pareto frontier, the PMBGA database would result in a poor ability to find commonality. In other words, if the database does not contain the needed information, then data mining will not recover it.

7.3.3 Experiment 3

The previous experiments reveal the ability of the clustering approach to capture component commonality. However, to help show how this approach could be used for a design product family flexibility study a different baseline design case is used. By using a different design study with different targets can test whether the clustering approach is generalizable. Additionally, the flexibility argument, sub-hypothesis 3.1, requires the clustering approach to yield an accurate partitioning for different performance requirements. This alternative baseline comes from Simpson, table 17[148].

Overall, the procedure for this experiment is similar to Experiment 1. This new design study also

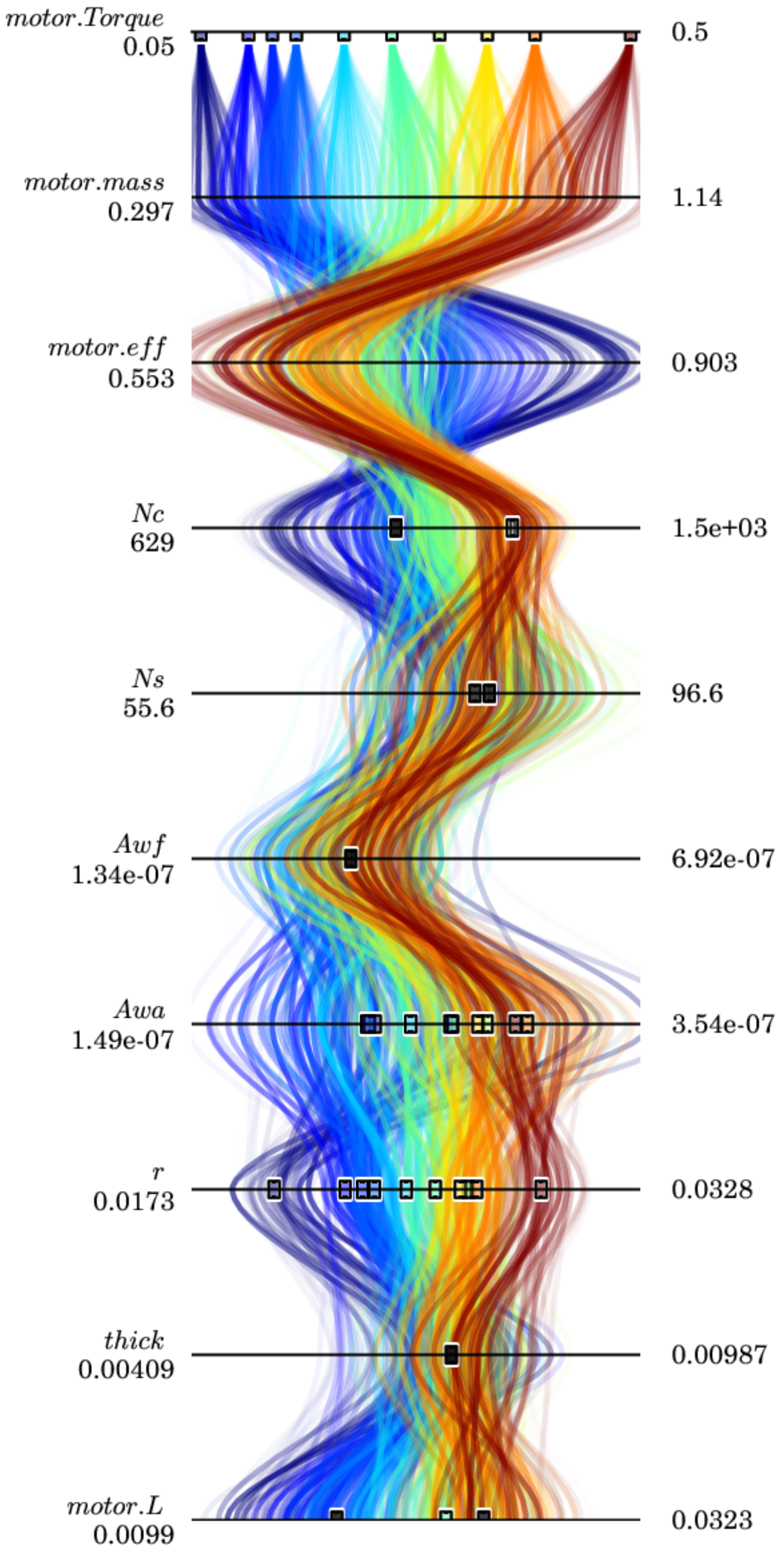


Figure 78: PMBGA Parallel plot

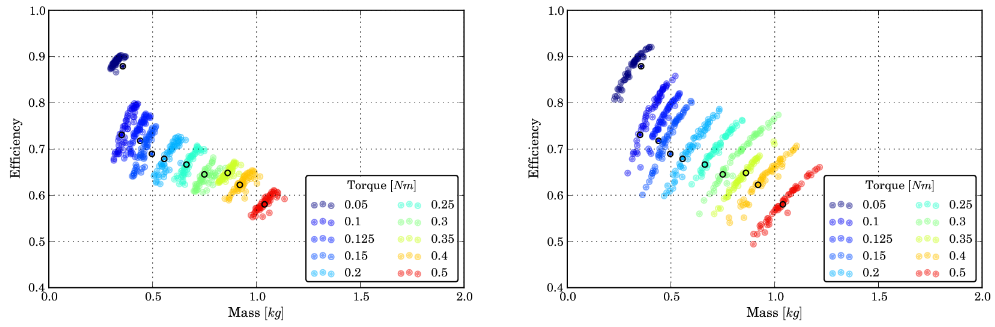


Figure 79: Comparison of Objective for the PMBGA

Table 17: Platform and Design Variable Settings from Simpson[148]

Motor	Design Variables								Objectives	
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L	i	η	M	T
1	1104	40	0.376	0.241	2.590	0.666	1.122	3.393	0.769	0.435	.05
2	1120	68					1.223	3.500	0.745	0.479	.10
3	1126	79					1.272	3.551	0.735	0.499	.125
4	1131	87					1.339	3.605	0.724	0.520	.15
5	1119	84					1.719	3.758	0.694	0.599	.2
6	1091	81					2.116	3.906	0.668	0.676	.25
7	1060	77					2.527	4.074	0.640	0.753	.3
8	1025	74					2.926	4.242	0.615	0.826	.35
9	987	71					3.333	4.421	0.590	0.898	.4
10	909	65					4.147	4.826	0.541	1.037	.5

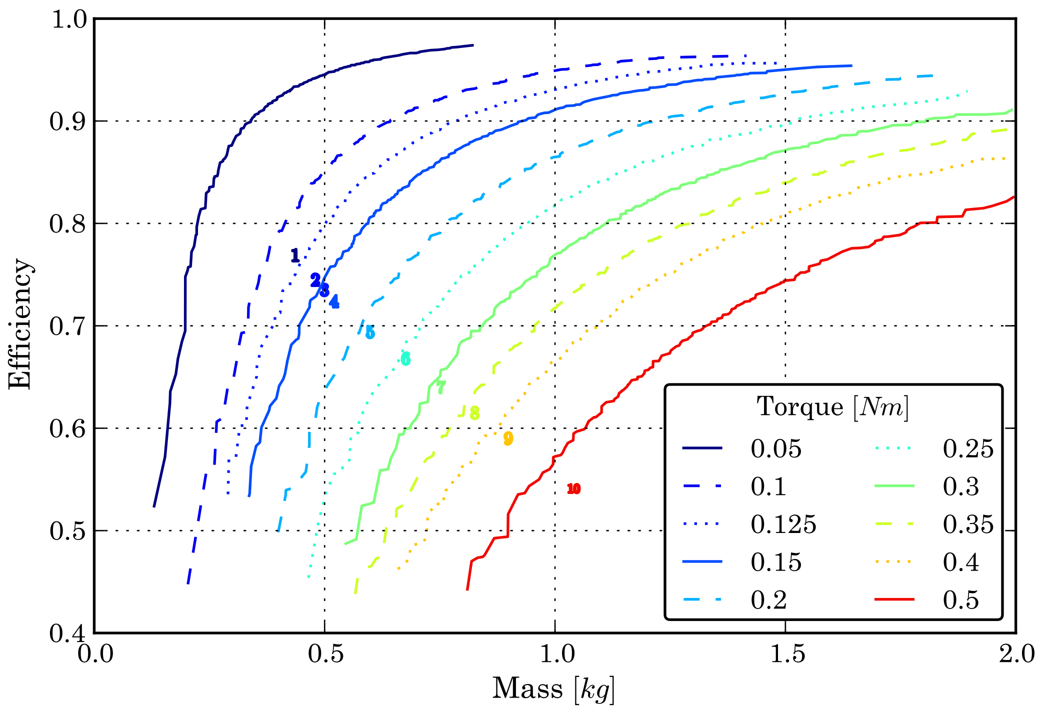


Figure 80: New performance requirements overlaid on Pareto Frontier of all Motors

demonstrates the consequences of having commonality across the entire set of products, figure 80. Each motor's performance takes a significant penalty when compared to its individually optimized Pareto frontier.

Using the alternative performance objective targets and the platform configuration, figure 81 shows the parallel coordinate plot of the feasible subset and its resulting fuzzy similarity relations. A few of these similarity relationships could be useful for the designer when selecting platform alternatives to evaluate. However, this experiment focuses on calculating the RRMSD to the baseline target values.

Table 18 shows the resulting design variable errors using this new baseline study. Overall, the RRMSD is lower then the original LHC or MC database in Experiment 1 baseline study.

Finally, the general database sparsity can be show in figure 82. This experiment used the database of 1 million points and, figure 83 shows the platform estimates to be widely distributed around the baseline targets.

Overall there are still very large estimates associated with this commonality approach. Perhaps

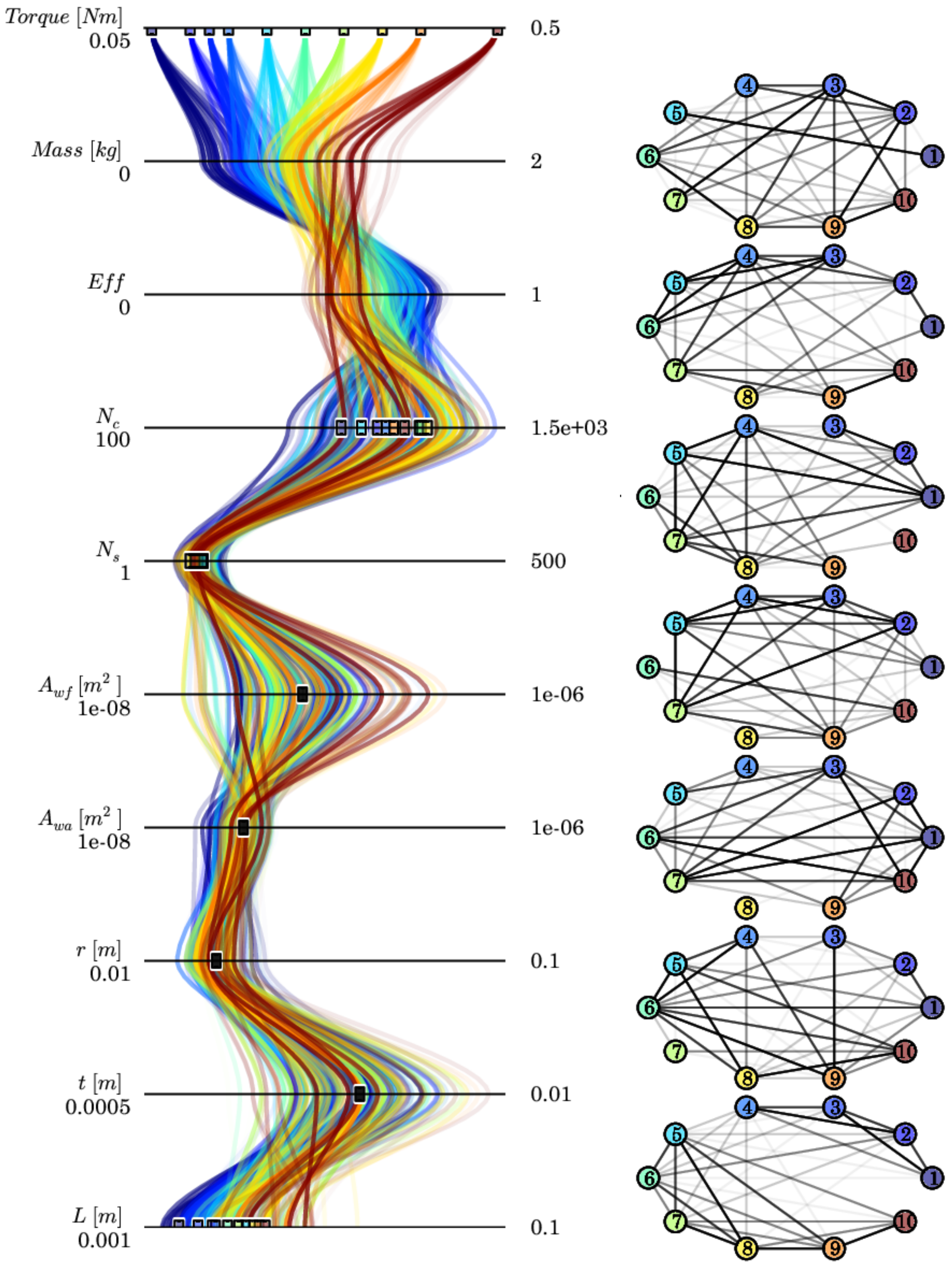


Figure 81: Experiment 3 Parallel plot using MC database

Table 18: RRMSD for Experiment 3

Motor No.	Design Variables						
	N_c	N_s	A_{wf}	A_{wa}	r	t	L
1	0.225	0.787	0.396	0.193	0.143	0.209	0.377
2	0.179	0.202					0.644
3	0.121	0.140					0.291
4	0.164	0.175					0.579
5	0.198	0.214					0.629
6	0.184	0.250					0.261
7	0.149	0.206					0.387
8	0.233	0.291					0.265
9	0.189	0.153					0.221
10	0.266	0.163					0.236

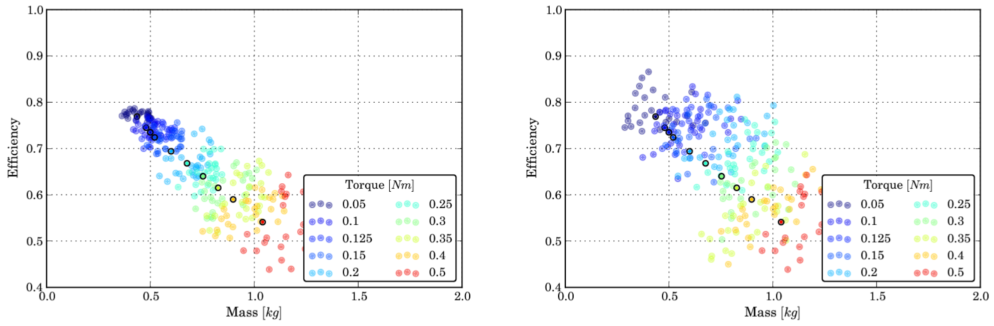


Figure 82: Experiment 3 Comparison of Objective

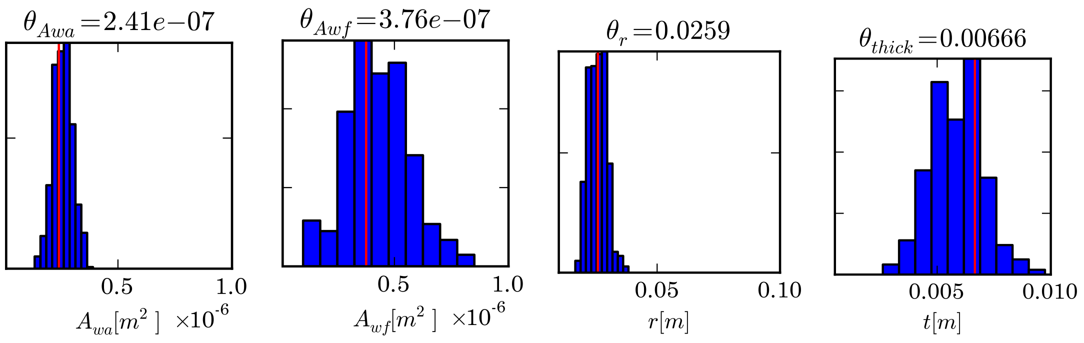


Figure 83: Feasible Subset Platform Histograms for Experiment 3

another DoE could be performed refining the design bounds, but it would result in excessive processing to make this approach competitive against other product family design methodologies. The next experiment tests the probabilistic approach's ability to capture component commonality.

7.3.4 Experiment 4

From Experiment 1, the baseline family is known to be near the Pareto frontier. Unless there is huge family cost savings that drives commonality, the family products will remain near their nondominated solutions. Experiment 4 tests the probabilistic approach using a Bayesian network trained to the electric motors' Pareto frontier. The first step is to find the Pareto frontier which has already been shown above with the PMBGA design space exploration. The next step is to use the Pareto frontier database to build a Bayesian network model. Figure 84 shows the Bayesian network structure found using a greedy search with random restarts. As expected the network is rather highly connected because the analysis model has all of these variables related.

The next part in building the network is to estimate each of the node condition densities. With this done the network has been trained to the joint probability distribution. This process is repeated for each of the ten motors in the family. The next step in surrogate modeling is to verify the model. Figure 85, shows the result from performing logic sampling for each of the electric motor Bayesian network models. Because there are no evidence nodes, a pure logic sampling is performed. Close comparison with the original data, figure 56, reveals the two structures to be very similar and supports the accuracy of the trained network models. Additionally, the objective space can be successfully recreated by sampling from the network, figure 86.

Figure 87 shows the resulting marginal distributions for each of the modules in the motor family. This distribution matrix is a useful breakdown of the overlap of likely component values for each product in the family.

With the now verified Bayesian networks, inferences can be performed to identify component commonality. First, the performance objective targets from the baseline family are used as evidence for each of the motors. Figure 88 shows the resulting posterior distributions for all of the modules in the motor family and generated by importance sampling. Because of an extremely high correlation between efficient and current, the current values are used as the performance targets. The

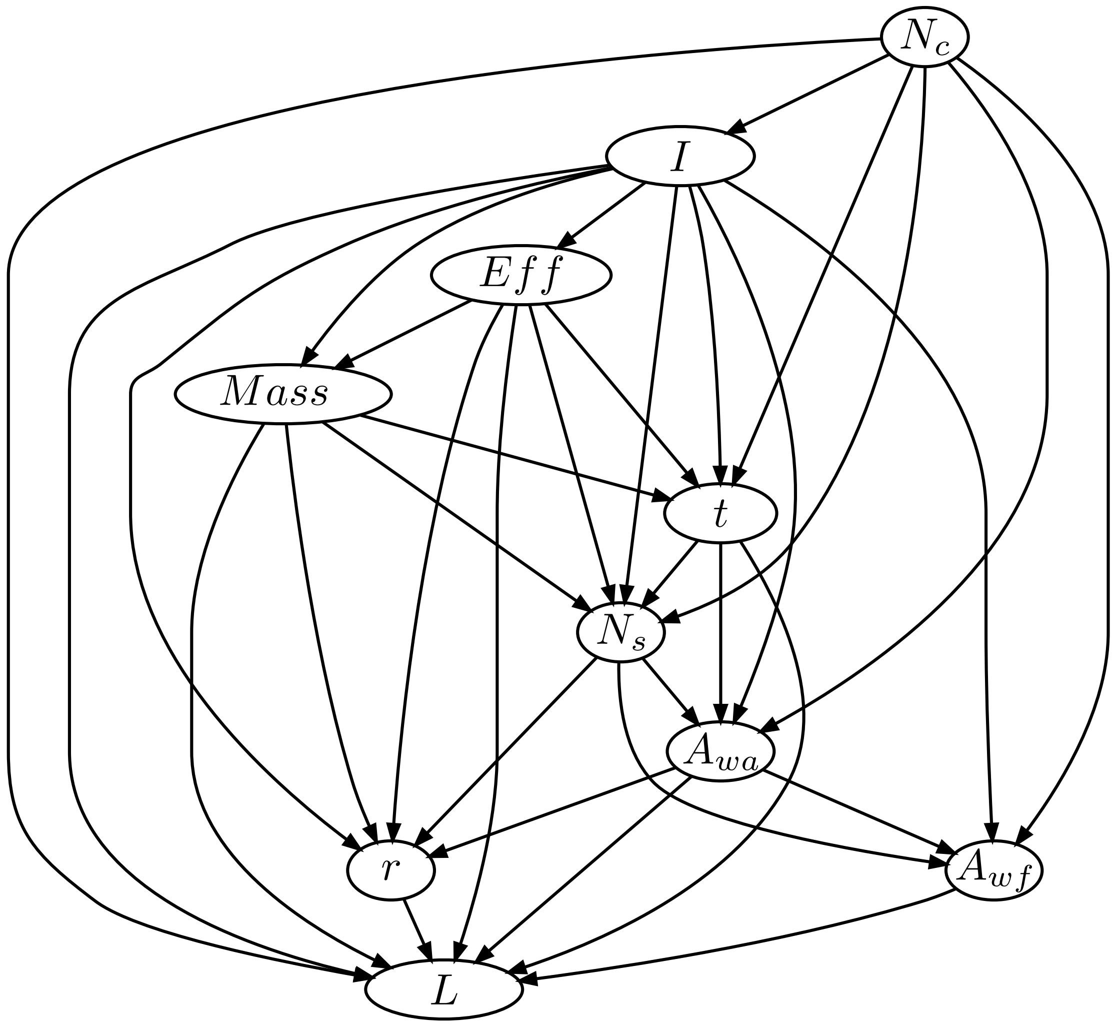


Figure 84: PMBGA Final Front Network Structure

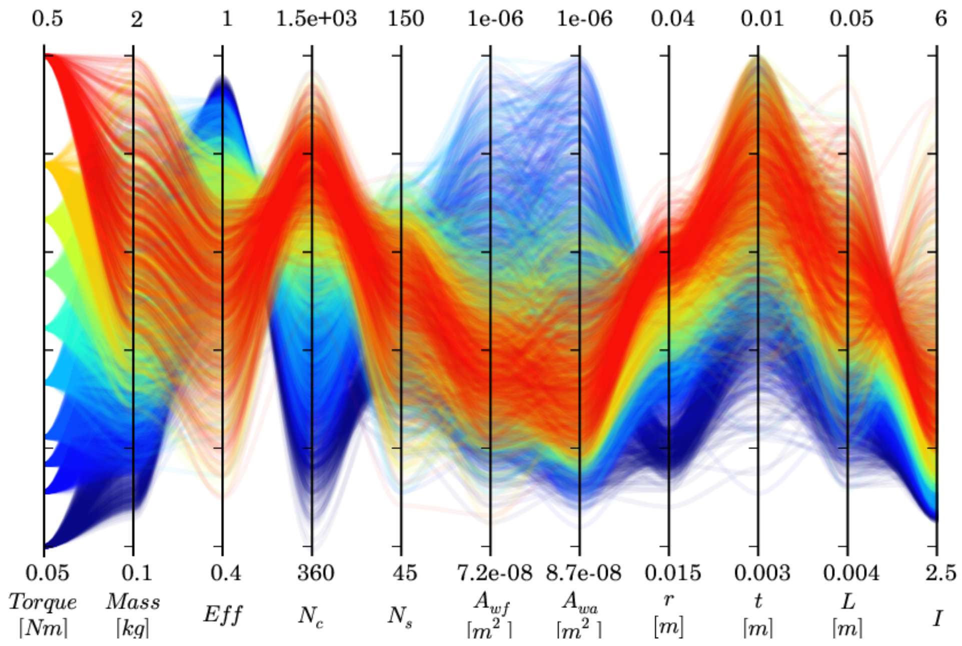


Figure 85: Sample from PMBGA Final Front database

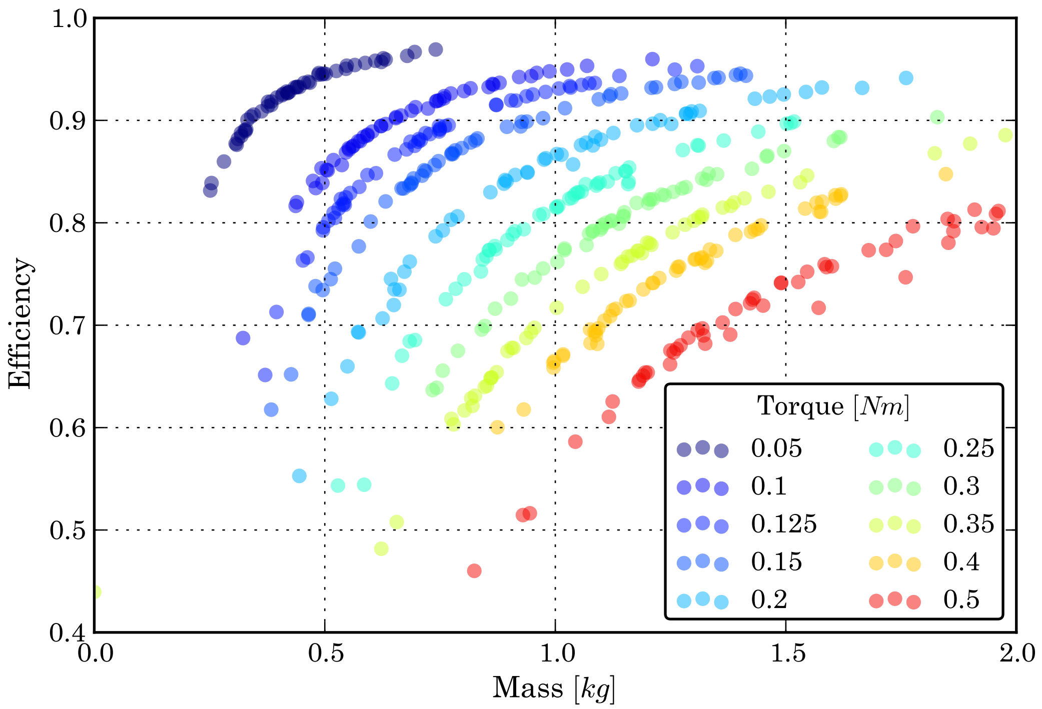


Figure 86: Objective Space Sample from Bayesian Network Models

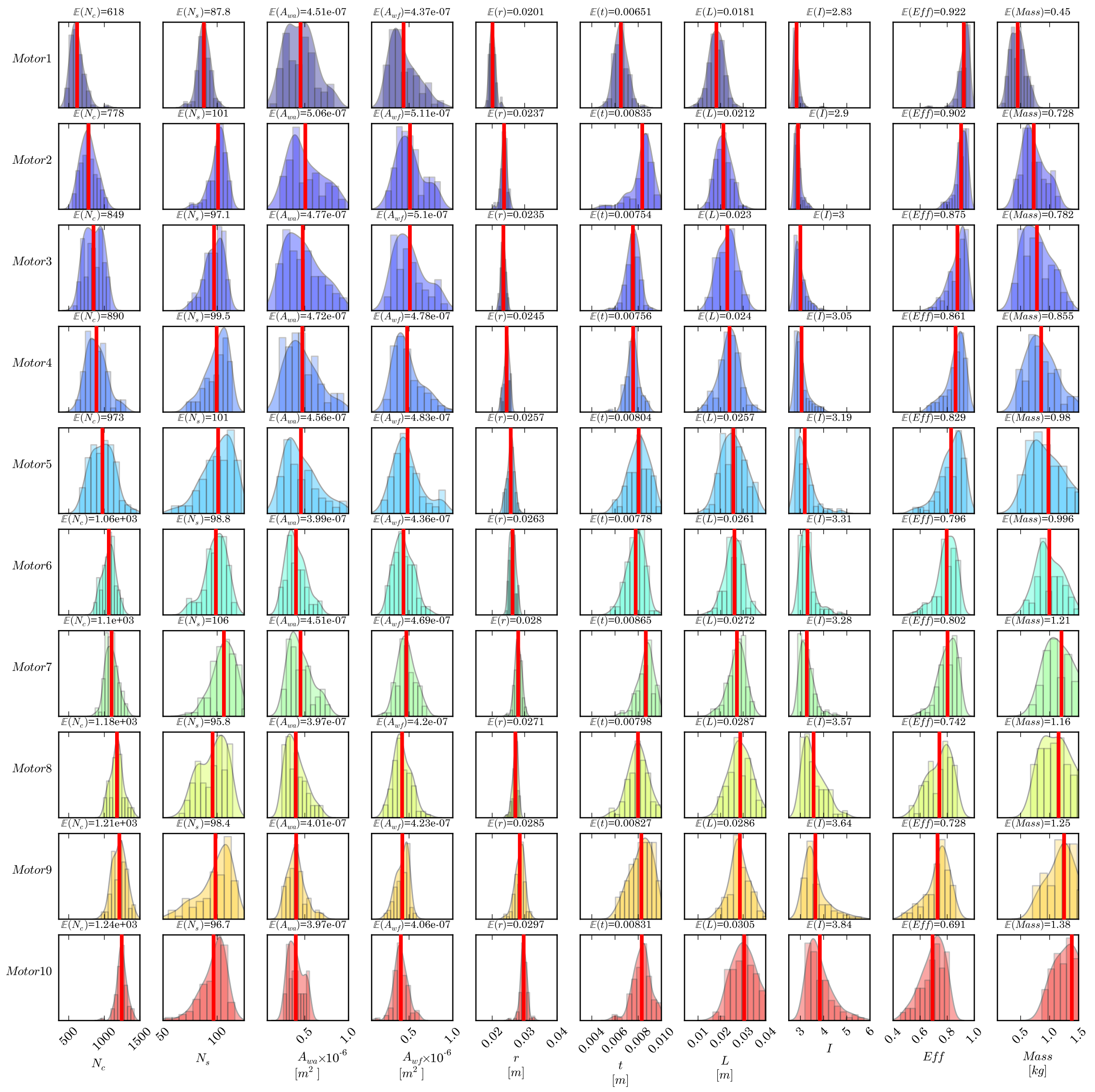


Figure 87: Module Marginal Distributions

similarity between two products in the module space uses these posterior design variable distributions to calculate the earth movers distance (EMD) between them. This similarity score can then be shown graphically in the weighted graph similar to the fuzzy similarity relations from the clustering approach. The darker edges in the graph indicate strong relationships between the components.

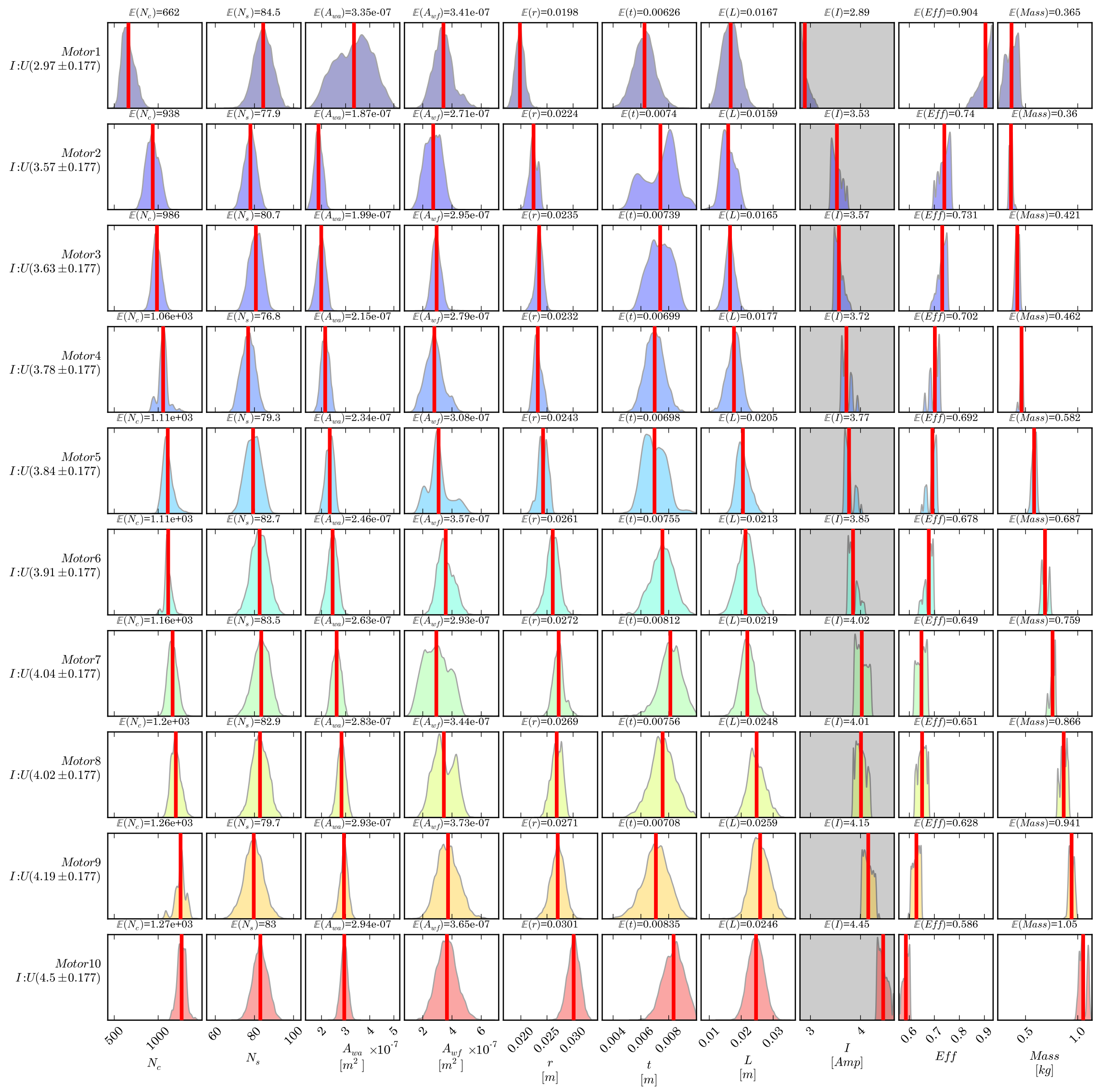
Compared to the fuzzy clustering similarity relations, this EMD appears to yield better and clearer representations of component similarity. The weighted graphs indicate strong overlap of all of the components for A_{wf} and t. Given a component partitioning and the motor posterior distributions, figure 91, shows the calculated platform posteriors distributions. The platform posterior distribution shows the likely values for the associated design variable. These posterior distributions can be applied as evidence for the next Bayesian network inference importance sampling.

Performing Bayesian inference with the motor performance targets combined with the above referenced platform posterior distributions yields new module posterior distributions, figure 92. These new module posteriors are processed again for similarity. The weighted graph structure shows a pretty clear partition for the L modules of the family. However, to compare with the baseline, the platform configuration will use {{1,2,3,4,5},{6},{7,8,9,10}} again. However, if this were an interactive design one may consider {{1,2,3,4},{5,6,7,8,9,10}}.

Using the component partitioning again, the platforming posteriors are found, figure 91.

With these new likely platform distributions as evidence, the Bayesian inference is repeated again, figure 92. The component similarities can be calculated once more to see if there are still any probability distributions overlapping. From the weighted graph, there appears to be some structure in the N_c and N_s modules.

Using the baseline platform partitioning for N_c and N_s yields new platform posterior distributions. Because of the knowledge encoded by the Bayesian network, the predicted platform distributions are much tighter to the target baseline design variables, figure 93. Finally to compare the probabilistic approach to the clustering approach, the importance sampling can be used to calculate the RRMSD, table 19.



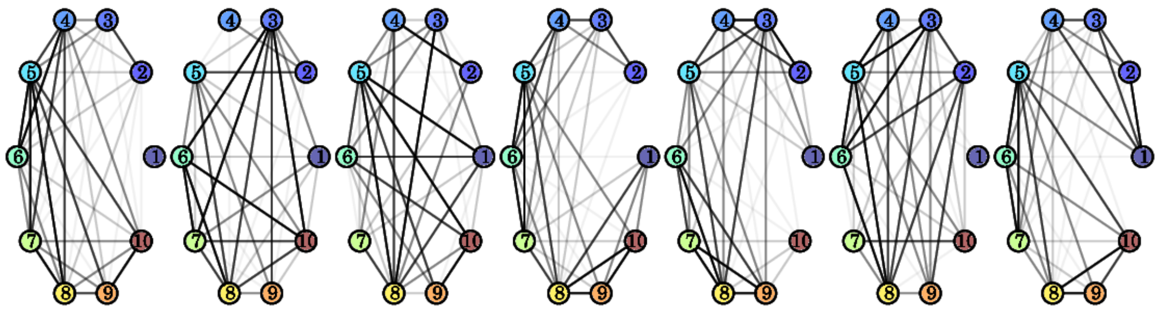
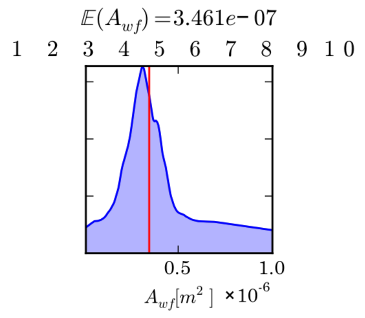


Figure 88: Module Posterior Distributions Inferred using Performance Targets



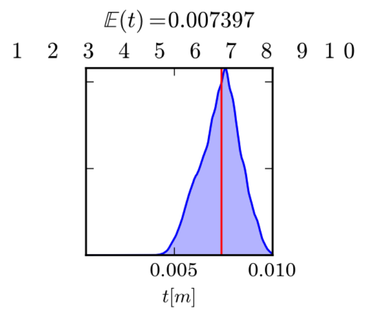
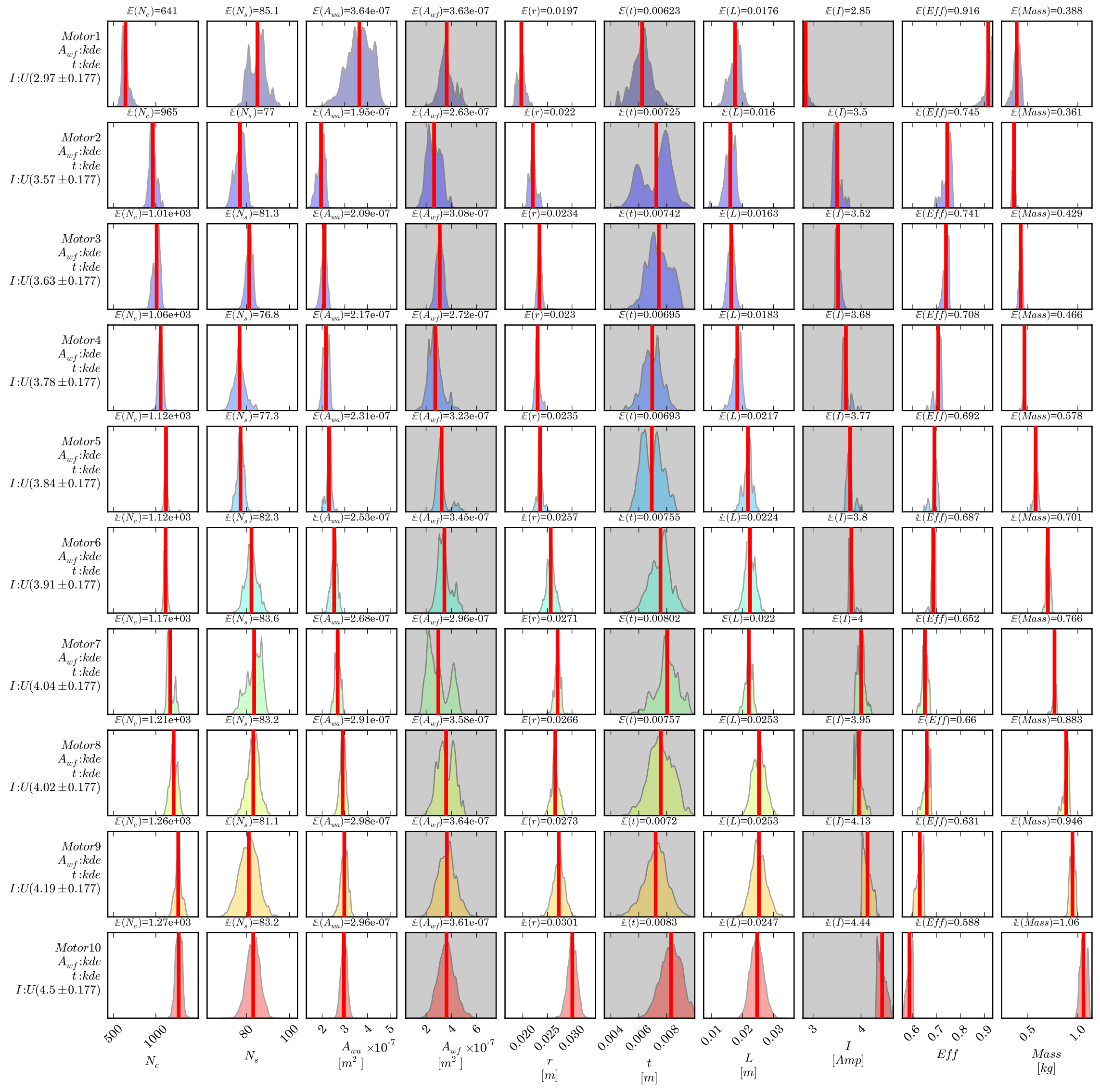


Figure 89: Platform Posterior Distributions

Table 19: Bayesian Network RRMSD

Motor	Design Variables						
No.	N_c	N_s	A_{wf}	A_{wa}	r	t	L
1				0.196	0.308		
2		0.259		0.059	0.207		
3				0.107	0.138		0.214
4	0.147			0.034	0.026		
5			.225	0.022	0.033	.125	
6				0.040	0.071		0.135
7		0.08		0.079	0.084		
8				0.041	0.082		0.120
9	0.033			0.035	0.060		
10				0.046	0.081		



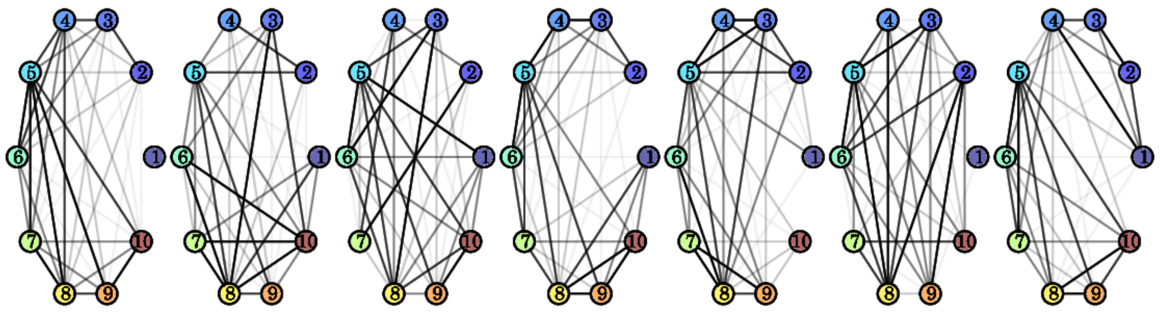


Figure 90: Module Posterior Distributions using A_{wa} and t platform posterior distributions and performance targets

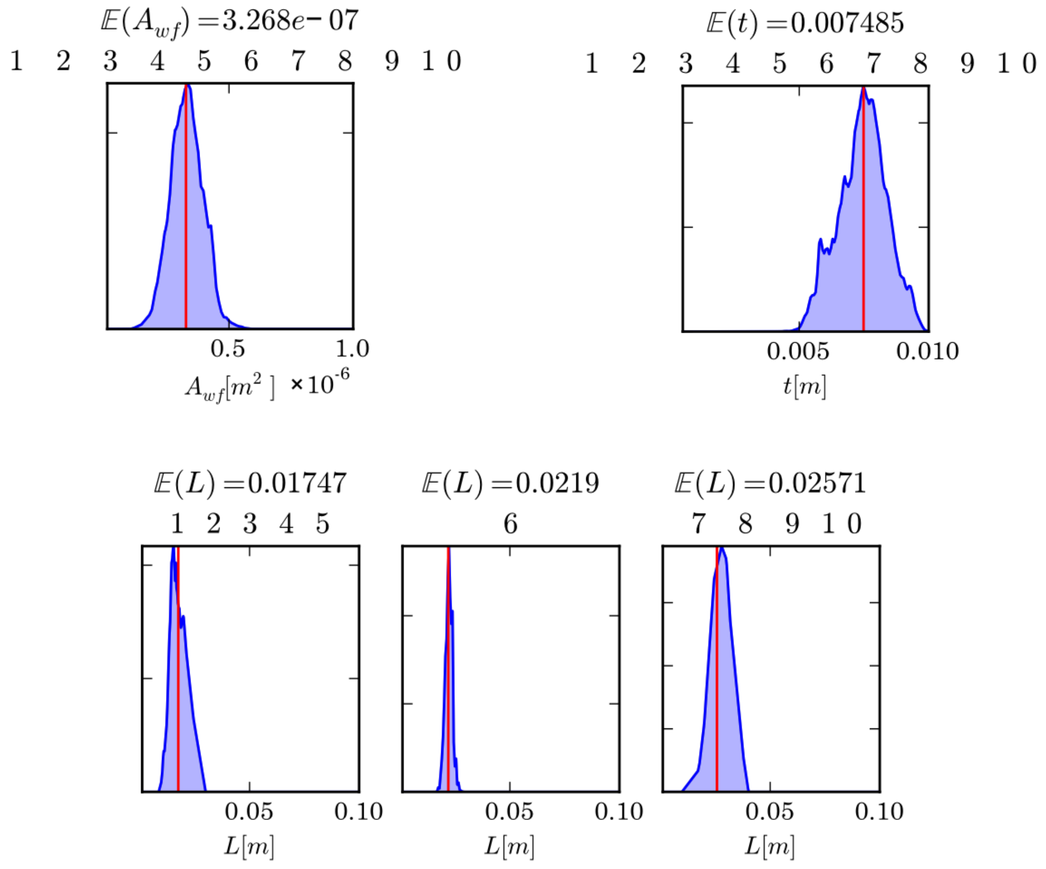
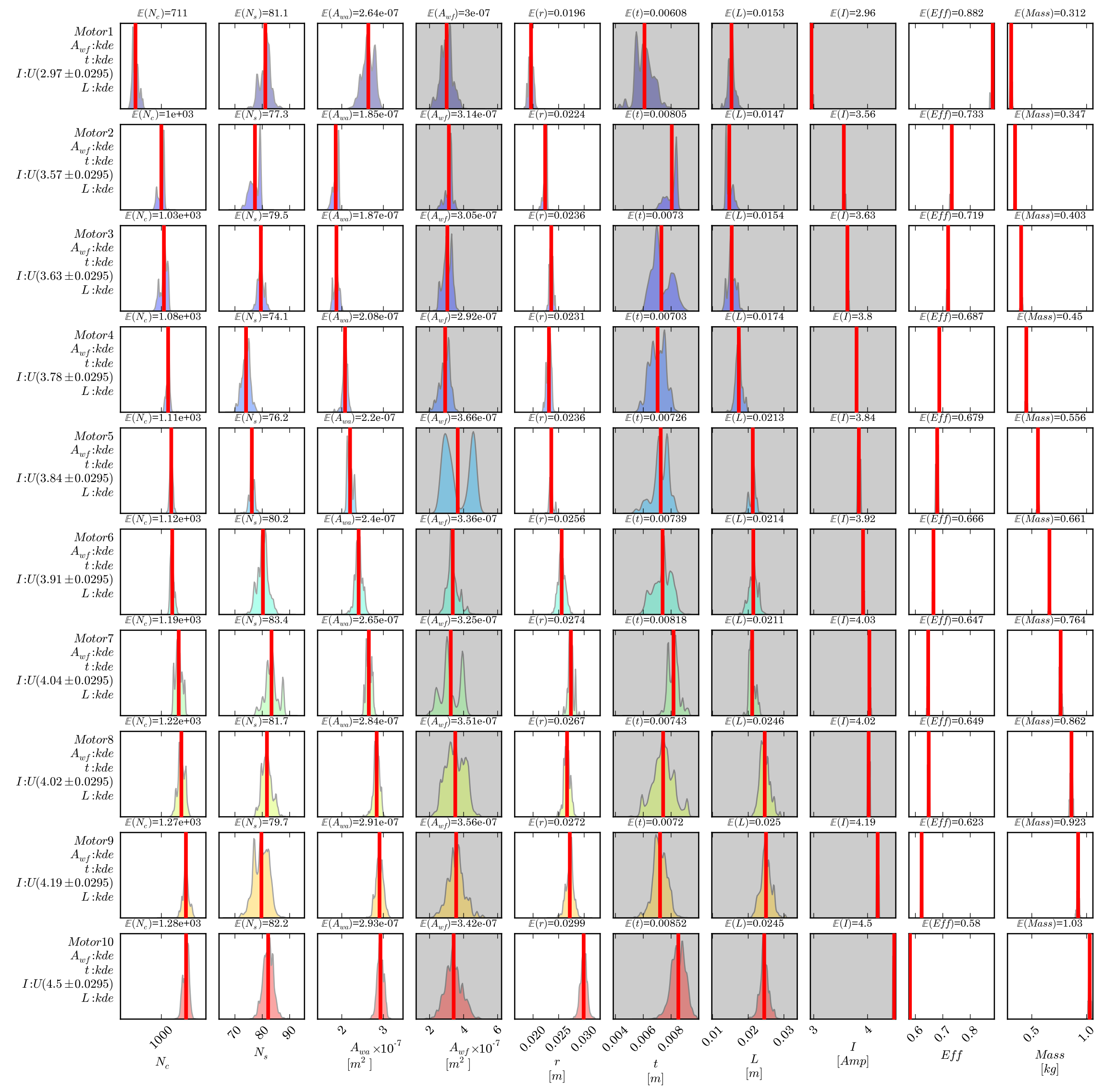


Figure 91: Platform Posterior Distributions for A_{wf}, t, and L



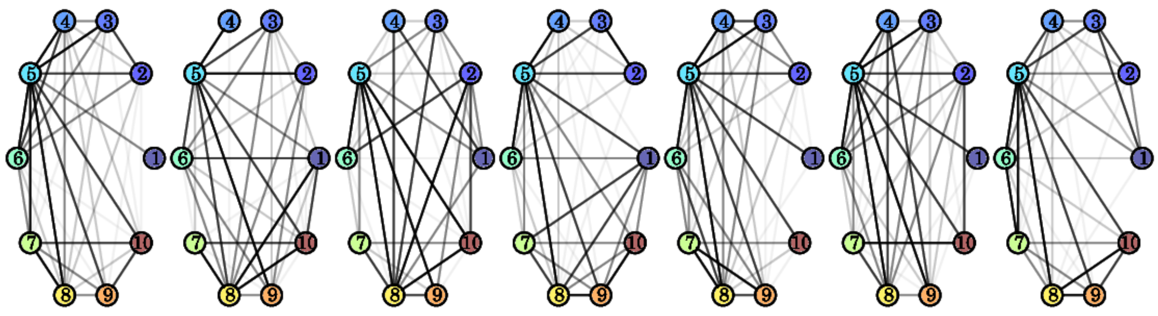


Figure 92: Module Posterior Distributions using A_{wf}, t, and L platform distributions

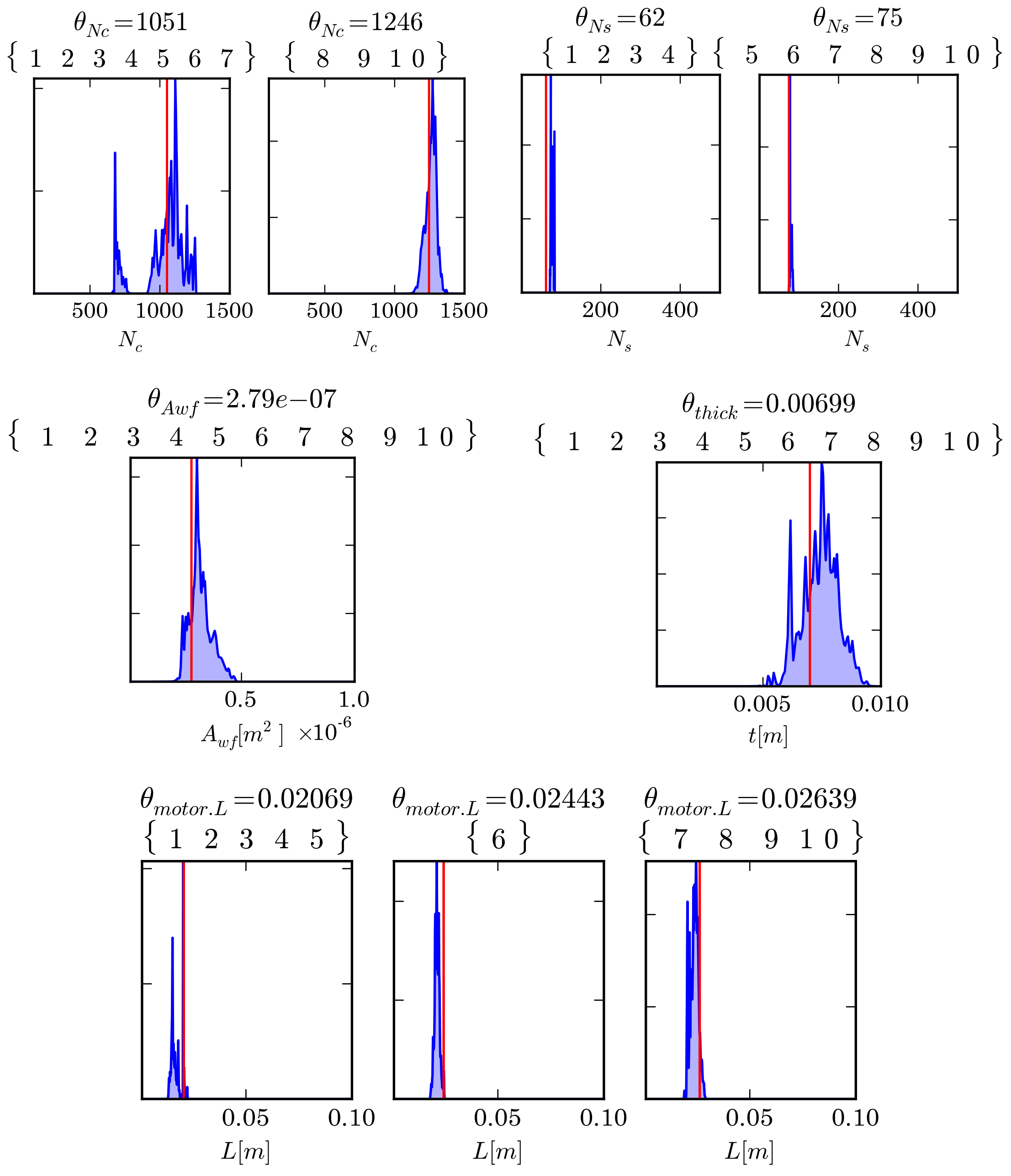


Figure 93: Posterior Platform Distributions using Bayesian Network Inference

Table 20: Method Comparison

	Average RRMSD
MOGA	0.270
LHC	0.434
MC	0.306
LHC Expanded	0.174
MC Expanded	0.183
Bayesian Network	0.137

7.4 Conclusions from Electric Motor Experiment

This chapter uses the universal electric motor problem to test the efficacy of the clustering and probabilistic commonality approaches proposed in this dissertation. This problem is particularly useful, because it has been used widely in other product family design studies, which makes it a useful benchmark for the proposed commonality identification approaches. It is important to benchmark these approaches on a known problem to calibrate their utility before extending them to a generic product family. As part of this comparison, the effectiveness of the predictions relies on selecting a feasible subset and quantitatively comparing it to the baseline values using the RRMSD. Table 20 is an average from the corresponding methods RRMSD and enables an easy final comparison between the approaches. Again lower values of RRMSD indicate that the subset is close to the baseline target.

Experiment 1 validates the ability to predict the platform design variables of the clustering approach. However, it also shows the estimates to be strongly linked to the size of the database. While it was shown that the equivalence hierarchy is similar to other studies, the platform estimate histograms can be wide, making it difficult to resolve component commonality groups. Refining the design variable ranges can improve the platform estimates.

Experiment 2 uses different design space exploration techniques with varying database sizes. The general trends again conform to the expectation that more cases and higher densities lead to better estimates.

Experiment 3 is an attempt to validate the clustering approach in a product family with flexibility requirements. Flexible products are capable of meeting alternative sets of requirements. By using a different electric motor product family design study with alternative performance objective targets and platform configuration, it is possible to assess the accuracy of the clustering approach. If the database is too sparse then the platform estimates are wide and the resolution is too poor for flexibility requirements sensitivity.

Together experiment 1, 2, and 3 show that while the fuzzy clustering approach can reveal the structure of component similarities, the platform estimates are wide. They can be improved with more cases but this may be prohibitive for anything but the fastest design codes. The fuzzy clustering approach does appear to be logically sound, if poorly suited to sparse databases. These experiments are designed to test sub-hypothesis 2.1 and sub-hypothesis 3.1, and to support the clustering approach as part of a larger product family design methodology. However, the sub-hypothesis 2.1 and sub-hypothesis 3.1 are difficult to accept because the number of design cases needed to be able to have a fine resolution with platform estimates is prohibitively large. Flexibility already assumes that performance requirement changes will occur over small ranges, and thata wide platform estimate will not be acceptable.

Experiment 4 tests sub-hypothesis 2.2 by taking the Pareto frontier of the baseline family from Experiment 1 to train a Bayesian network. This experiment shows an ability for the Bayesian network to be trained to a midsized design problem with 10 dimensions and accurately recreate the Pareto frontier. The experiment also finds that the platform prediction errors are much more accurate. This experiment supports the use of a continuous Bayesian network to capture the joint distribution of the design space and to perform inferences that adhere to the Pareto frontiers. Furthermore, this experiment supports the use of the probabilistic approach to perform module commonality identification.

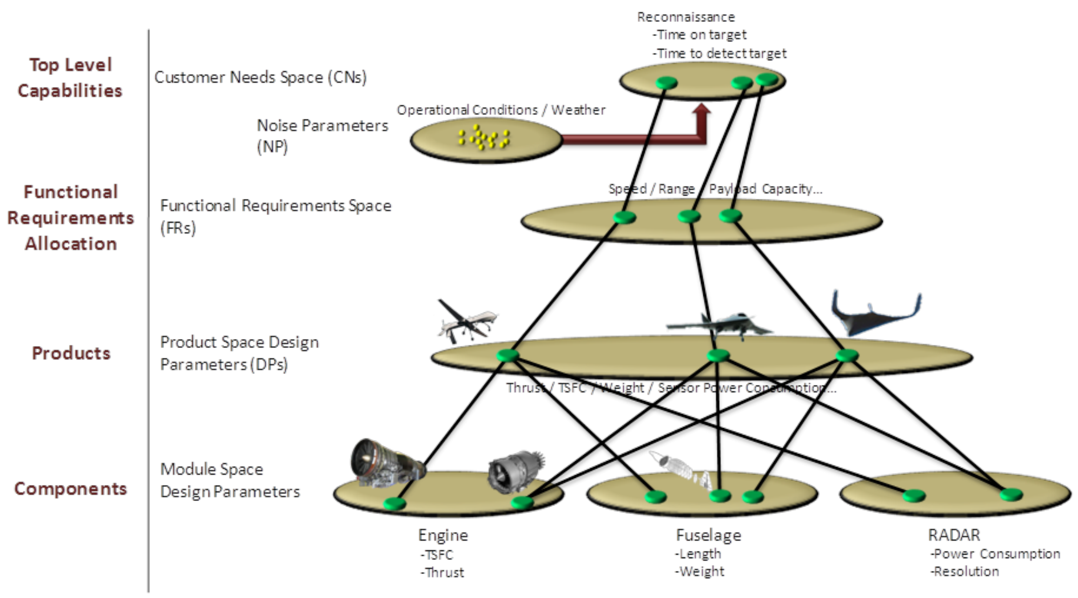
Chapter VIII

DESIGN OF AN AIRCRAFT FAMILY

This chapter presents a product family design study of an aircraft family operating as a system-of-systems across two scenarios. One scenario uses unmanned aircraft to perform a set of maritime monitoring missions while the other uses unmanned aircraft to perform an aerial firefighting mission. These scenarios can take advantage of commonality opportunities to create a useful product family. This is a Class III product family problem because there is neither a prior configuration knowledge , nor an existing library of components. However, this problem is more complex than other Class III problems, because the aircraft act cooperatively to achieve the system-of-systems customer needs.

Additionally, this chapter also serves to demonstrate elements of the generic product family design methodology. This serves to further test the effectiveness of the two commonality identification methods, the fuzzy clustering approach, and the probabilistic approach, which are also tested in the previous chapter. Observations can be made about their practical ability to guide product family design using an aircraft family with no existing knowledge or constraints.

To help understand the aircraft product family a breakdown showing hierarchy between the different domains in the aircraft product family is shown in figure 94. On the figure, the lowest domain is the individual module spaces for each component in all of the products. In this example, only the engine, fuselage, and radar is shown but in reality there can be many components considered. The next higher domain is the product space where each product is composed of all of their shared components as well as any unique components. Using the various physics-based performance models, the products' performances are evaluated and shown in the functional requirements space. Examples of performances that belong to the functional requirements space are vehicle speed, payload capacity, and sensor detection range. Finally, the highest domain is the customer needs space. At this level, it is possible to see how well different sets of products are able to fulfill important customer needs, like price and effectiveness in minimizing damage from fires. The mapping from



Top Level Capabilities
Customer Needs Space (CNs)
Reconnaissance
-Time on target
-Time to detect target
Noise Parameters (NP)
Operational Conditions / Weather
Functional Requirements Allocation
Functional Requirements Space (FRs)
Speed / Range / Payload Capacity...
Products
Product Space Design Parameters (DPs)
Thrust / TSFC / Weight / Sensor Power Consumption...
Components
Module Space Design Parameters
Engine
-TSFC
-Thrust
Fuselage
-Length
-Weight
RADAR
-Power Consumption
-Resolution

Figure 94: Product Family Domains Diagram

functional space to the customer needs space can be done with various types of SoS models.

8.1 Establish the Need

The first step of the generic product family design methodology is to articulate the customer's needs. In this case, the customer wants a set of aircraft that can perform in two unique scenarios. The first scenario (called "the maritime monitoring scenario") is a set of unmanned aircraft the patrolling the Norwegian maritime Exclusive Economic Zone (EEZ). The EEZ extends 200 nautical miles from the Norwegian coastline and is a complicated environment with a plethora of commercial and private vessels. It falls under the responsibility of the Norwegian Coast Guard to safeguard the mariners at sea and enforce maritime laws and sovereignty.

Figure 95 shows an actual map of the vessels off the coast of Norway using the Automatic Identification System (AIS). Each of the colors represent a different type of vessel[2]. The map illustrates high densities of vessels in certain areas near the coast. Because the maritime environment is so heavily traveled, it is inevitable that maritime accidents will occur. To that end one major concern for this scenario and the region in general is the protection of the lives of the sailors. The aircraft must be capable of assisting in any search and rescue operations which includes having an aircraft capable of identifying persons in the water as well as another able to drop aid.

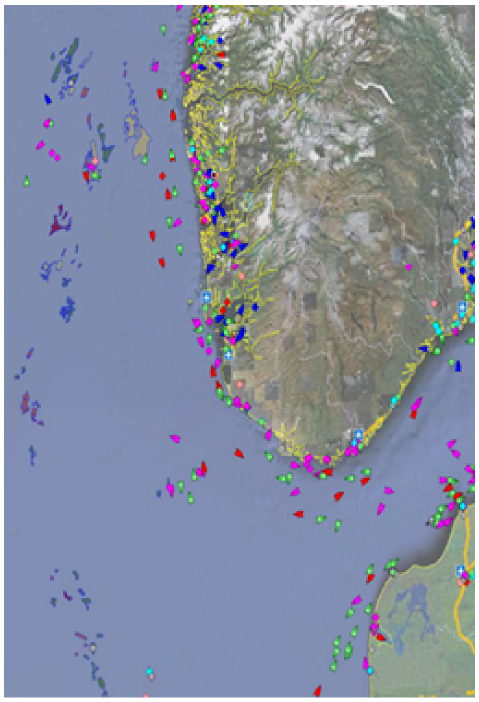


Figure 95: Norwegian Coast

Much of Norway's economy relies on aquaculture and its offshore petroleum industry. Moreover, these aspects of Norway's economy are particularly vulnerable to environmental degradation. Such ecological damage may be caused by many elements of local industry, most importantly overfishing and, particularly, illegal oil discharge. One study in a similar maritime environment demonstrates that much of the oil discharged comes from numerous small intentional events[116]. Oil discharge may occur several ways such as discharge of ballast water, tank washing, and engine room effluent discharge [116]. Protecting the maritime environment from illegal and unregulated fishing as well as illegal oil discharge from shipping is vital to sustaining the economy.

To maintain the safety and health of this region there must be a mechanism in place that is capable of monitoring the illegal discharge of oil slicks, and the required dispersion and clean-up of these slicks. Moreover, pursuant to the customer need to protect the lives of sailors in the region, this mechanism must also monitor the safety of distressed vessels in need of rescuing. Currently there are several manned assets that perform these missions, but in the future there is an opportunity for unmanned aerial vehicles to perform these duties.

The second scenario (called "the firefighting scenario") is the monitoring and mitigation of fires

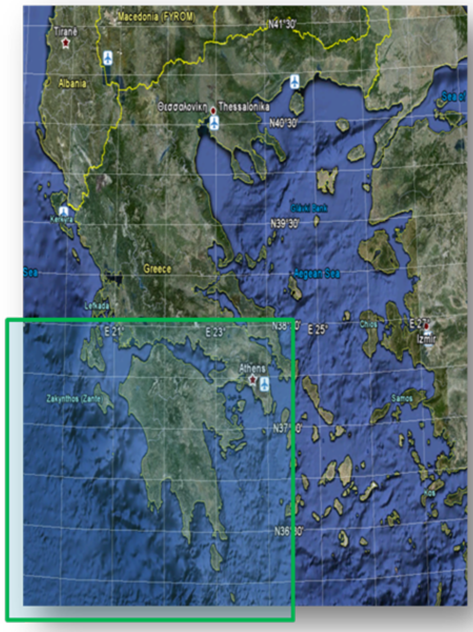


Figure 96: Area of Greece under Consideration

on islands in the Aegean Sea. Housing of local firefighting equipment is both expensive and difficult because the islands in the Aegean are geographically disperse with large distances between then and limited infrastructure. Thus, with hot summers, dry conditions, and human negligence, combined with high winds make fires on these remote mountainous islands a real threat.

Real time surveillance with high altitude aircraft is particularly useful to meet the customer need of minimizing fire damage. When the fires do appear, these surveillance aircraft provide continuous coverage of the site as aerial tankers drop fire retardants. Here, there is also the opportunity for unmanned aircraft to perform these missions, since these systems are always vigilant. Figure 96 shows the a map of the Aegean used to develop SoS model.

Using these identified customer needs, a functional breakdown is performed. In this case, the functional breakdown describes, at a high level, the missions or tasks that need to be performed to satisfy the customer needs. For the functional breakdown, background research identifies six missions for the maritime monitoring scenario and two missions for the firefighting scenario. For the maritime monitoring scenario the missions of interest are: (1) patrol of fishing and oil platforms, (2) ship tracking, (3) environmental monitoring, (4) oil spill response, (5) search for distressed vessels, and (6) rescue and provide aid to sailors on distressed vessels.

The fishery patrol and ship tracking missions help to provide maritime domain awareness to the Norwegian Coast Guard in its task of maintaining security and enforcing sovereignty. Environmental monitoring is also critical to the protection of the Norwegian coastline from maritime polluters who dump bilge and wash oil tanks in the open sea. The primary goal of this mission is to detect and verify oil slicks while collecting sufficient evidence for legal action. Once an appropriately large spill is detected, there is a need to respond. The oil spill response mission needs to be capable of delivering the means to quickly combat the slick, such as oil dispersants to breakup the slick before it interferes with wildlife.

In addition, occasionally there are ships that require assistance. The aircraft needs to be able to search for and locate distressed vessels and monitor the situation until additional assistance arrives. Once finding the distressed vessels, a rescue mission is executed to deliver emergency aid, such as a life raft.

In the firefighting scenario, the missions considered are: (1) fire monitoring, and (2) firefighting. In the fire monitoring mission, aircraft mustprovide expansive coverage of the fire prone land. Once a fire is detected, a firefighting mission is performed to drop fire retardants that aid in extinguishing the fire.

Table 21 shows the different requirements for each of the identified missions. It is possible to see areas of overlap between the different missions that can serve as a basis for the family architecture.

The similarities and differences of these two problems are clear. Both of these problems exhibit strong need for a persistent patrol, which is better conducted at higher altitudes where aircraft benefit from improved fuel efficiency. Additionally, these scenarios need vehicles with low altitude capabilities to deliver time sensitive support. The use of aerial vehicles in both of these mission types is not common, but there is an opportunity to develop unmanned aircraft family for these missions[44, 90].

In summary, customer needs for the aircraft product family are identified through background review. In the maritime monitoring scenario, the customer needs are to locate possible oil spills and patrol Norway's EEZ. Examples of some resulting MoEs to determine how well these needs are met include how long it takes to detect an oil spill, and whether the polluter is caught spilling. For the

Table 21: Mission Specific Characteristics

Mission Requirements	Maritime						Firefighting		
	Environmental Monitoring	Ship Tracking	Patrol	Search	Oil Spill Response	Rescue	Patrol	Fire Monitoring	Firefighting
Long Loiter Times		X		X				X	
Thermal Imaging	X			X		X		X	X
RADAR	X	X	X	X	X	X	X		
Long Range	X	X	X	X			X		
Payload Delivery					X	X			X
Time sensitive				X	X	X			X
Rough Weather Conditions		X	X	X	X	X	X		
Harsh Environment								X	X
Hover over a station						X			
BLOS Communication	X	X	X	X	X	X	X		

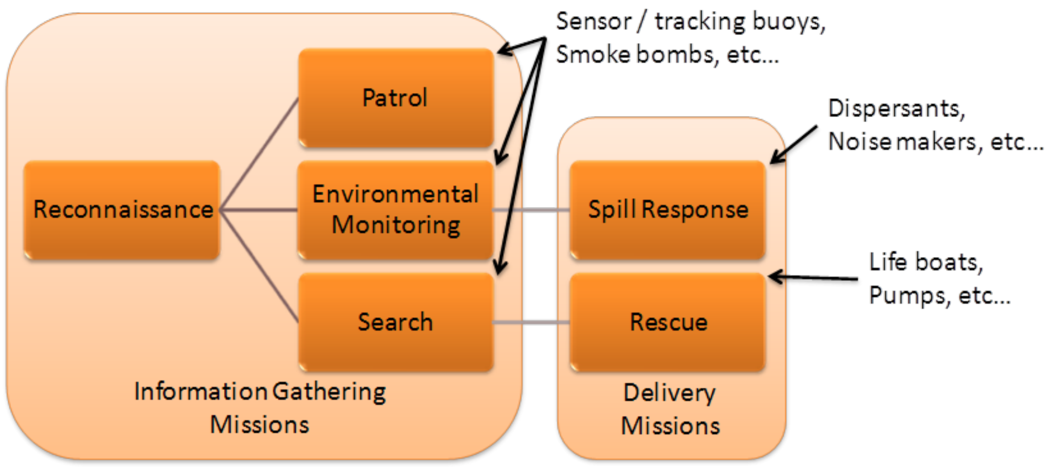

Sensor / tracking buoys, Smoke bombs, etc...
Dispersants, Noise makers, etc...
Patrol
Reconnaissance
Environmental Monitoring
Spill Response
Life boats, Pumps, etc...
Search
Rescue
Information Gathering Missions
Delivery Missions

Figure 97: Mission Hierarchy

firefighting scenario, the MoE includes the total burnt land and system cost. These two scenarios require a stochastic SoS model to understand the effectiveness of potential aircraft in meeting the customer needs. This problem also has potential family benefits from sharing aircraft components. Because this is a Class III product family problem, the next step is to formulate alternative family architectures.

8.2 *Define Family Architectures*

The appropriate allocation of different missions to particular aircraft is one way to help define family architecture. In the maritime scenario, the missions are grouped into two types: information gathering, and time sensitive delivery, figure 97. The reconnaissance mission is treated as a superset of the patrol, environmental monitoring, and search missions. Information gathering missions also trigger the delivery missions of spill response and rescue once their respective targets are found.

After inspecting the mission hierarchy, different high level family architectures are developed. Figure 98 shows the first architecture being considered. Here, there is one aircraft that performs all of the information gathering missions and a different aircraft that is called to perform the special delivery missions. Simple mission profiles are included for each of the different missions and are used to help develop aircraft mission performance models.

The second architecture that could be considered is shown in Figure 99. Here, the information gathering missions are combined with their unique delivery segments. For example, for the search

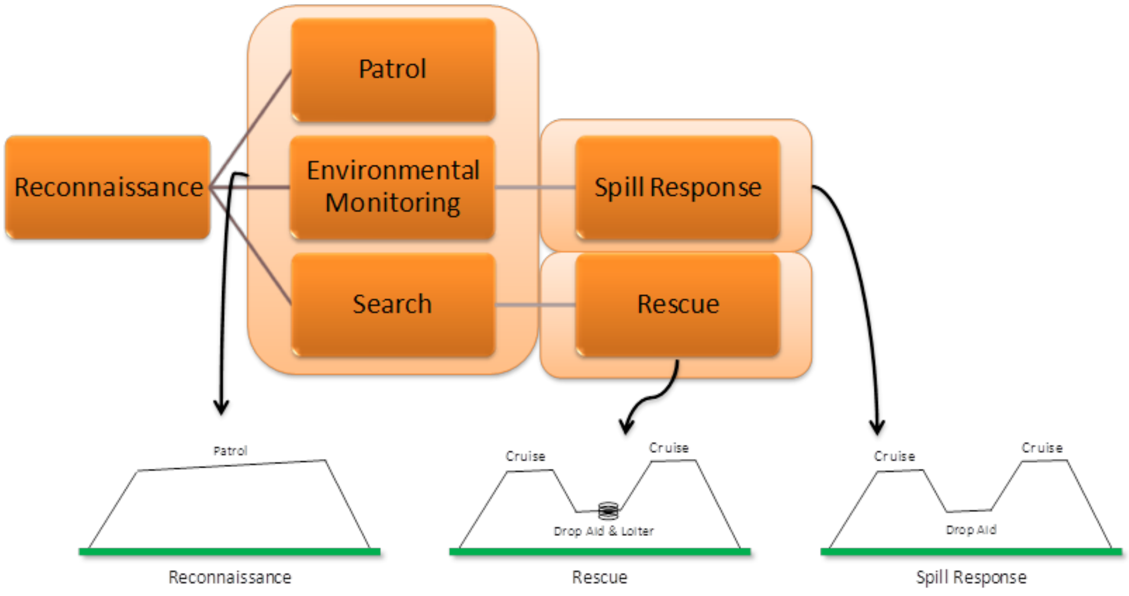


Figure 98: Maritime Architecture One

and rescue missions, one aircraft performs the searching and then is capable of delivering aid to distressed vessels.

The firefighting scenario only has two missions: reconnaissance, and delivery of a fire retardant. Again, there are two main ways of combining these missions: having one aircraft searching and dropping, or having unique aircraft types for both roles.

Given no existing information or preferences, it is not clear which family architecture would be superior to meeting the customer needs at the lowest costs and best family commonality. In a full design study, models for these unique architectures would need to be developed to evaluate the alternatives.

For further development of this study, and to focus on testing the component identification approaches, the main family architectures that are considered are the ones that separate time sensitive dropping missions from the long endurance, persistent missions. However, the models created will be flexible enough to allow for alternative architectures to be evaluated in the future.

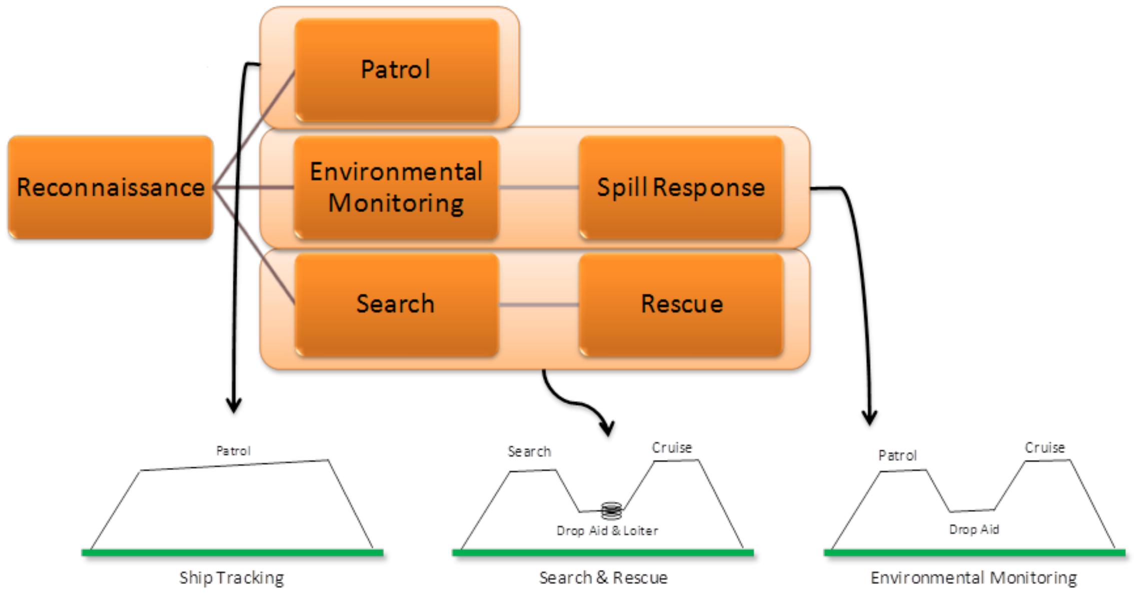


Figure 99: Maritime Architecture Two

8.3 Establish Value Objectives

At this point in the family design process, a family solution has been architected that can meet the customer needs. This step helps to establish the specific importance of the customer needs in relation to each other. Without these preferences, it is not possible to arrive at a final ideal family. For this demonstration problem, there have been a number of measures of effectiveness (MoEs) that describe the aircraft family's overall performance. In the maritime monitoring scenario the overall MoEs are the family costs, the time it takes to search and rescue, the time to detect an oil spill, the time to cleanup the oil spill, and, finally, the probability of identifying the polluter. The MoEs for the firefighting scenario are the size of the burnt land and the cost of the family.

There are four aircraft types in the family architecture. While the ideal commonality is not known between the aircraft components, it is important to identify the components that should be considered for commonality, i.e. $\mathbb{S}^{pq}$. To simplify the analysis, the aircraft in both scenarios have a traditional tube and wing configuration and are assumed to be roughly a Medium Altitude Long Endurance (MALE) UAV. The aircraft are modular so that they share some common geometry components like the fuselage and wings.

The fuselage component is characterized by a length and diameter design variable. However, it is possible to consider only the diameter and add fuselage plugs for the different aircraft fuselages as needed. Though the wing module can be described using many parameters, this study focuses only on the wing area and aspect ratio.

The engine is also considered a significant component and is considered as a possible family platform. The engine module can also be parametrized with many different design parameters depending on model fidelity. For example, a high fidelity model of a turbofan engine would enable the engine to be parametrized by internal geometry and materials. This kind of model would allow building an engine platform based on core or bypass subcomponents. However, if a low fidelity model is used it can be sufficient to identify engines based on the required thrust setting.. The low fidelity model is preferred in circumstances where constraints on time or computational resources limit model complexity.

Finally, in both scenarios there is a need for a strong sensor suite so that the radar modules will also be analyzed. In this case, the radar module space will be characterized by the radar detection range. Higher fidelity models and interactions in the SoS model allow the impacts of radar operating frequency or other similar parameters to be traded.

The choices in objective values and module spaces are not trivial and have far reaching impacts. For instance, the cost objective requires an associated cost model while the firefighting scenario requires understanding how the size of the burnt land is calculated. Changing the objectives or altering the model space parametrization may require updating the modeling and simulation environment if those variables are not satisfactorily included in the initial models.

8.4 Generate Feasible Alternatives

To be able to generate feasible alternatives, a suitable modeling and simulation environment needs to be created. This environment is driven by models that capture the module level design parameter interactions and impacts on the MoEs.

8.4.1 Modeling and Simulation

This demonstration design study requires a complex modeling environment to be created to generate the design database which captures everything from the component level to the customer needs

Table 22: Comparisons of ABM and DES

	Agent Base Model	Discrete Event Simulation
Pro	Better visualization of environment	If the targets are randomly distributed then they can be considered as Poisson random variables . Because of this, the time between the events are exponentially distributed
	More intuitive in setting up the physics of the environment.	Efficient use of computer time by jumping to next queued event
Con	Agents must be programmed with appropriate level of intelligence. IE search patterns, reactions etc	Difficult to understand transition probabilities in complex sequences of events, environmental factors hard to encode, interactions between different processes
	Fixed discretization of time and space	Tactics can be hard to implement if there is not a clear sequence of events that are followed

level. To be able to capture the customer needs level, a system-of-systems (SoS) model needs to be created. There are several methods to model and analyze SoS each with their own strengths and weaknesses. The most common methods for exploring these complex design spaces are discrete event simulations, and agent based models (ABMs).

To investigate the trade-offs of these two methods, simple test search models are developed using both techniques. From these tests, a list of benefits of each of the two methods can be determined and compared to the objectives of this research. Table 22 lists of the pros and cons of the two methods.

ABM simulations are the best option for this problem. They are more flexible to changing operational behaviors once the base agent logic is coded. If the MoEs included operational availability or maintenance metrics then a discrete event or combined simulation would be needed. The full integrated modeling and simulation environment, includes several system level models. For example, sensor models and an aircraft sizing model act as inputs for agent capability metrics of the ABM which then calculate their mission effectiveness.

8.4.2 Integrated Simulation Environment

The development of the integrated simulation environment is the source of ongoing research. This section reiterates previously published work on testing integrated simulation environments. [125].

The sizing and synthesis of aircraft for this integrated environment requires several varying code disciplines. Because of the similarity between the two scenarios, it is possible to reused analysis models between the two environments. This helps model development by lessening the duplication of effort. For this exploration, the aircraft are modeled in FLOPS using the notional medium altitude long endurance aircraft baseline. From this baseline the dimensions are parametrized so that many different sizes of aircraft can be generated driven by the payload requirements.

Figure 100, is an image of the modeling and simulation environment for the firefighting scenario. The appropriate initial inputs are selected from the design of experiments, which is combined with each aircraft, and sized independently, depending on the needs of the payload volume and the sensor package. Following the aircraft sizing, the SoS simulation is executed.

Two sensors are considered on the aircraft: radar and IR. Both the radar and IR sensor models are developed from simple physics and implemented in Matlab. In the maritime monitoring scenario, radar is used to detect ships at sea, while for the firefighting scenario, radar is used to detect fires by the return from smoke. For the radar modeling, the basic radar equation is implemented which is based off of first principles.

IR sensors are used to identify the polluters in the maritime monitoring scenario. In the firefighting scenario, the IR sensor can reveal additional details about the fire. The IR sensor model implementation is from[95]. In an initial sensitivity study, the IR sensor was dominated by all of the other components. Therefore, to help limit the number of variables in the model, the IR sensor model is removed as an independent component.

The volume sizing of the aircraft captures the dependency of the geometry and payload volume. With this volume sizing approach the internal placement of each of the components is ignored. This is a conceptual design study it is valuable to find the sub-class of vehicles that are capable of completing the mission. In the future higher fidelity models that can capture the impact of the internal component arrangements can be included. These models could be helpful to distinguish between alternative family platform configurations. Because, internal geometry plays an important role in the ability to modularize the aircraft. The interfaces between alternative components needs be standardized but these detailed questions are better addressed after the conceptual design phase. For the volume sizing, the aircraft configuration selected is a simple tube and wing, and that the

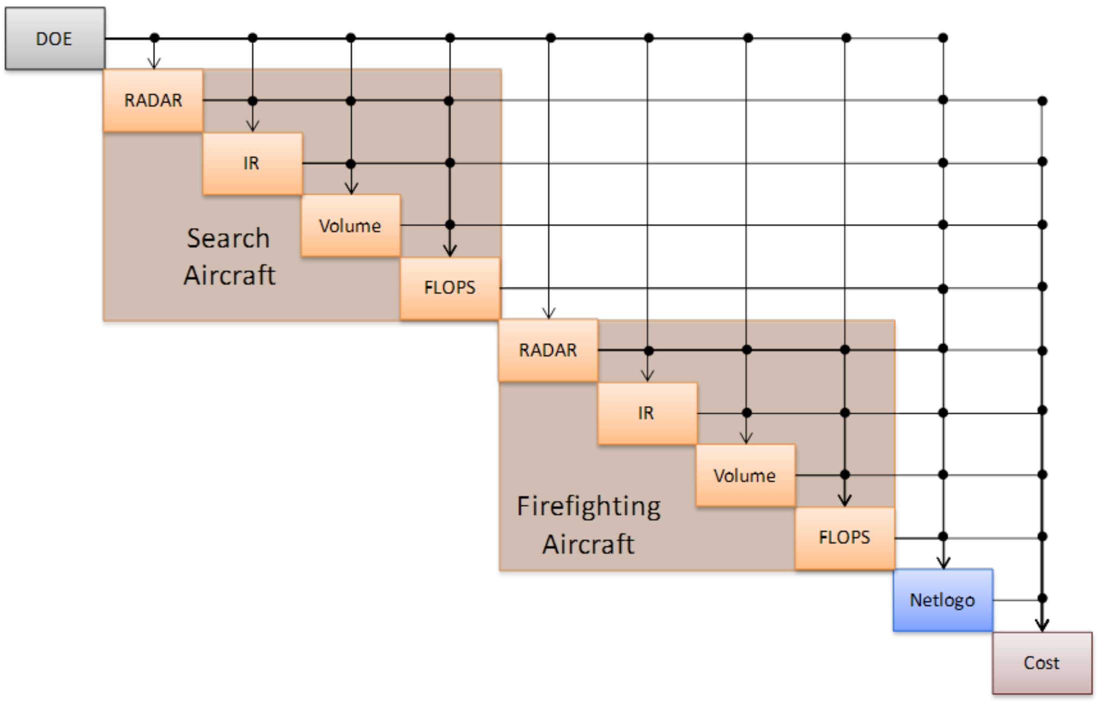


Figure 100: Integrated Modeling and Simulation Environment

avionics and other internal components will not increase or decrease in volume from a baseline aircraft. These two assumptions imply that the changes to the fuselage length directly impact the size of the payload the aircraft is carrying capable. The inputs for this volume sizing method are the payload volume, the percentage of the payload carried internally, and the geometry from the MALE UAV. The outputs are the fuselage length and the payload weight.

The aircraft sizing is conducted using a code known as FLOPS with some of the input geometric coming from a simple volumetric sizing tool also made in Matlab. The goal of these models is to make the aircraft as parametric as possible to help explore the design space and look for possible component commonalities while still having aircraft capable of meeting the customer needs.

Once the fuselage geometry is determined, it and the weight estimates for the sensors are then used by the FLOPS model to size the two aircraft. As mentioned before, the sizing of the aircraft is conducted on a rubber aircraft from a state of the art MALE model. Each scenario has a search aircraft and a drop aircraft with slightly different mission profiles resulting in four unique FLOPS models. All of the aircraft models use a turboprop engine deck and are based on a rubberized engine model for a MALE aircraft. The FLOPS model is used to calculate optimal mission conditions,

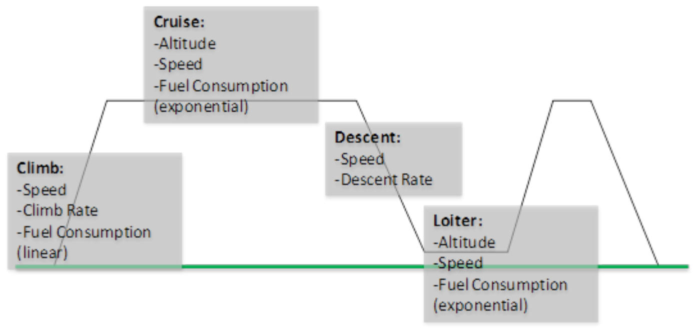


Figure 101: Flight Profile of Drop Aircraft

components and gross weights. The FLOPS model outputs the engine size as well as the wing geometry, speed, and fuel consumption characteristics.

After the search and drop aircraft are sized, they can then be evaluated in the SoS ABM simulation environment. The ABM takes: high level variables like the numbers of aircraft to operation, as well as sensor performances like detection range. Additionally, the ABM takes detailed aircraft flight profile parameters, figure 101, from the FLOPS model to accurately represent the aircraft while performing the scenario missions. The stochasticity enters into the ABM through the fact that mission events occur throughout the geographical areas. For instance, the distressed vessels for the search and rescue mission can be anywhere in the EEZ or the initial location for the fire can happen anywhere on land.

The logic, shown in figure 102, is similar for both SoS simulations and shows the robustness of the agent behavior implementation. First, the simulation is initialized and a task manager allocates aircraft to the needed missions. An example of this allocation is in the firefighting scenario. Once a fire is detected, the manager identifies an aircraft that is capable of delivering the needed retardant payload. Within the simulation only a subset of the aircraft are capable of completing this mission, and thus one of the drop aircraft is selected to complete the mission in lieu of the searching aircraft. A check is also performed of the aircraft's ability to successfully complete the mission. Currently, this check determines whether or not the aircraft has enough fuel to complete the mission. If it does not, a different aircraft is selected. The benefit of ABM is that the base agent behavior rules can be updated to account for different analysis mission scenarios.

After all the tasks have been assigned to proper assets, the simulation time is incremented.

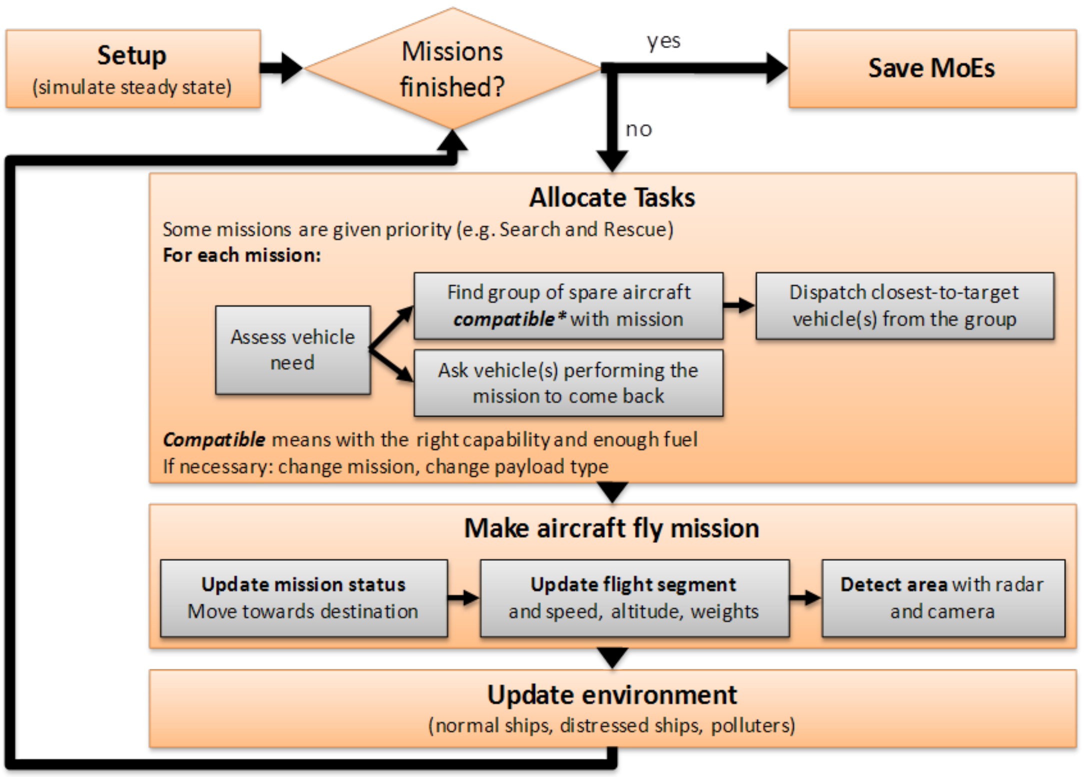
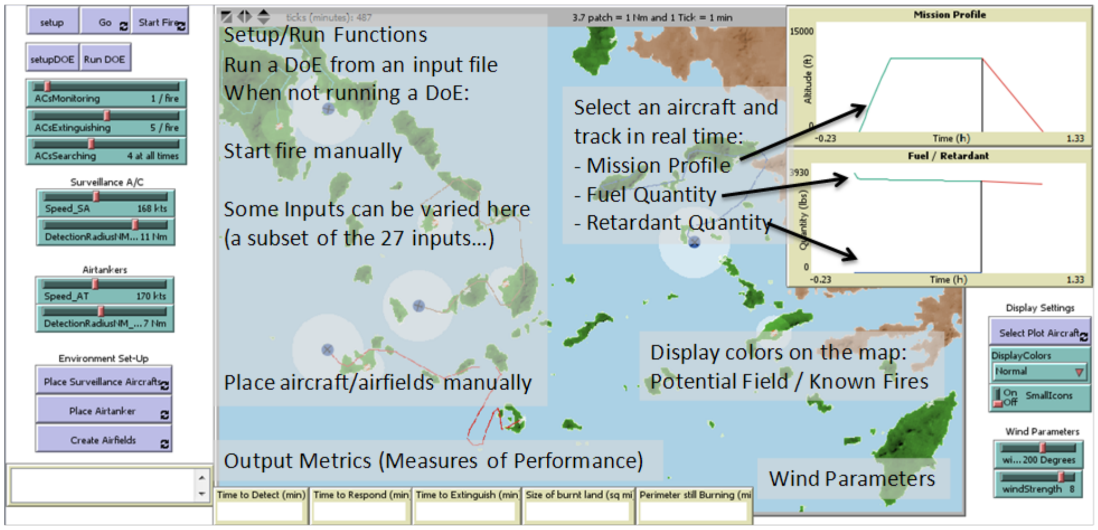

Setup
(simulate steady state)
Missions finished?
yes
no
Save MoEs
Allocate Tasks
Some missions are given priority (e.g. Search and Rescue)
For each mission:
Assess vehicle need
Find group of spare aircraft compatible* with mission
Dispatch closest-to-target vehicle(s) from the group
Ask vehicle(s) performing the mission to come back
Compatible means with the right capability and enough fuel
If necessary: change mission, change payload type
Make aircraft fly mission
Update mission status
Move towards destination
Update flight segment
and speed, altitude, weights
Detect area with radar and camera
Update environment
(normal ships, distressed ships, polluters)

Figure 102: Logic of Simulating Environment[125]

Aircraft fuel burn is calculated for this time increment as well as updates to other dynamic agent properties. For instance, in the firefighting scenario, the aircraft payload changes as the retardant is dropped on the fire. Consequently, this will lighten the aircraft and improve its fuel economy. Additionally the retardant affects the fire and has partially prevented its spread.

This process of incrementing time and updating the agents continues until an exit criteria has been reached i.e. the fire extinguished or the maximum simulation time is exceeded. The only difference between the two scenario simulations in terms of task allocation is in the maritime scenario. Search and rescue mission allocation takes precedence over the oil slick identification and dispersant missions. This is because the possibility of saving lives is most important.

An image of the simulation environment interface can be seen in figure 103. This interface is a representation of the firefighting scenario. The map of the islands can be seen in the center. This map is used to identify which areas of the simulation are capable of hosting a fire. Additionally having a visual interface allows the designer to playback interesting combinations of design variables and watch the resulting mission capability unfold.



ACsMonitoring 1 / fire
ACsExtinguishing 5 / fire
ACsSearching 4 at all times
Speed_SA 168 kts
DetectionRadiusNM... 11 Nm
Speed_AT 170 kts
DetectionRadiusNM_... 7 Nm
DisplayColors Normal
On Off SmallIcons
wi... 200 Degrees
windStrength 8

Figure 103: Aerial Firefighting ABM Frontend[125]

One important logic feature needed for realistic aircraft behavior is the control loop to search the map. Fixed patrol patterns can be given but these are rigid and not flexible to changing events. The specific patrol algorithm for this simulation has been developed by ASDL to search the area in a systematic yet flexible matter[125]. For this search function each pixel is set to a cost function for each of the searching aircraft. The cost function indicates the last time the pixel was seen, the closeness of the pixel to other aircraft and the closeness of the pixel to the specified aircraft. This cost function encourages aircraft to explore areas which have not been seen recently as well as prevents different aircraft from searching the same region.

An image of the cost function can be seen in figure 104. The small circle in the image represents the the radar detection range of the aircraft while the large circle shows the cost function. The darker colors show an increased need to search that area. Areas that have recently been searched by the aircraft are lighter in color while the area that has not been searched, is very dark.

8.4.3 Model Verification

Now that the simulation environment has been implemented, it is important that the environment is a reasonable representation of reality. It is not possible to validate this modeling and simulation environment because of its large scope and its use of agent based models. Additionally there does not exist a current baseline system performing these scenarios to calibrate the models against. However

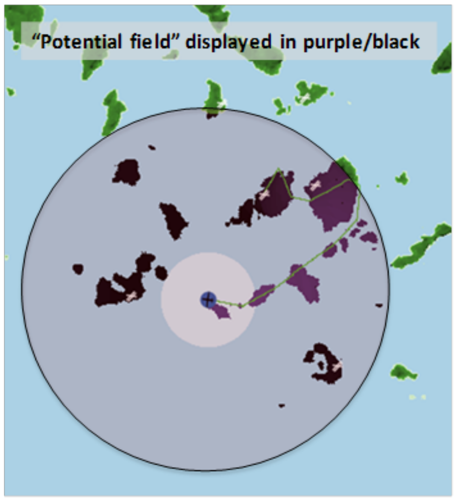


Figure 104: Searching Algorithm Cost Function Visual Representation[125]

it is possible to verify at least that the model has trends that follow reality and that will serve as a useful basis for performing family trades.

Table 23 shows the design variables and their corresponding ranges for each aircraft type being considered. A Latin Hypercube design of 60,000 cases was run for the firefighting scenario while a 120,000 design was used for the maritime scenario. The ratio of failed cases for the maritime scenario was about twice that of the firefighting scenario so the resulting database is roughly the same number of successful designs.

The main trends that need to be verified for the firefighting scenario is the size of the burnt land. One technique is to fit a neural network to the LHC successful cases. This neural network then is useful for creating visualizations that should display the appropriate trends. Figure 105 shows three contour plots where the amount of burnt land is shown for different combinations of aircraft types. The left most contour shows the baseline case. The middle contour plot shows that increasing the radar detection range reduces the influence of the number of search aircraft on the resulting size of the burnt land. This is because the search aircraft is more capable and detects the fire sooner so the air tankers can respond faster. The contour plot on the right shows the result of increasing the amount of retardant carried by the tanker aircraft. This verifies that a more capable drop aircraft significantly reduces the overall burnt land. This figure as a whole supports the general verification of the major trends on the aerial firefighting modeling environment.

Table 23: Aircraft DOE Ranges

Variable		Low				High			
		Fire Drop	Maritime Drop	Fire Search	Maritime Search	Fire Drop	Maritime Drop	Fire Search	Maritime Search
Mission	Number Delivery Aircraft	2				8			
	Number Search Aircraft	1				6			
	Range	400 nmi	900 nmi	600 min		842 nmi	1342 nmi	1800 min	
	CruiseMach	0.207				0.33			
	Altitude (ft)	10000	5000	10000	-	30000	20000	30000	-
Fuselage	dispersantVol(gal)	0		-		1000		-	
	Cargo_Distribution	0.189473684		-		1		-	
	growthVolume ft ^3	0				25 ft			
Wing	WingArea (ft^2)	180				1190		400	
	AspectRatio	6				19			
	TaperRatio	0.72				1			
	ThicknessChordRatio	0.11				0.17			
Engine	Thrust (lb)	3000				8500			
Radar	Frequency (GHz)	8				18			
	Antenna Diameter (m)	0.5				1.5			
	Range (m)	37040				74084			
	Average Power (w)	100				600			

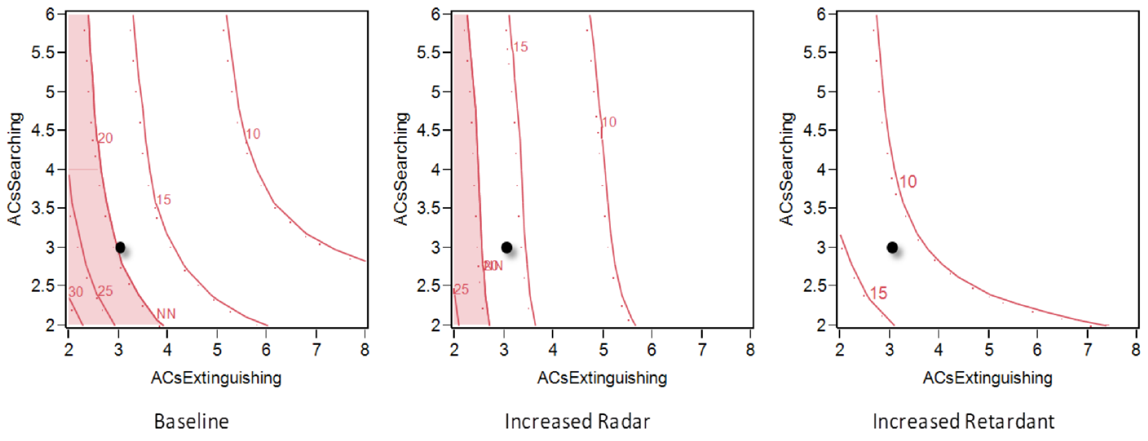


Figure 105: Firefighting Neural Network Contour Plot

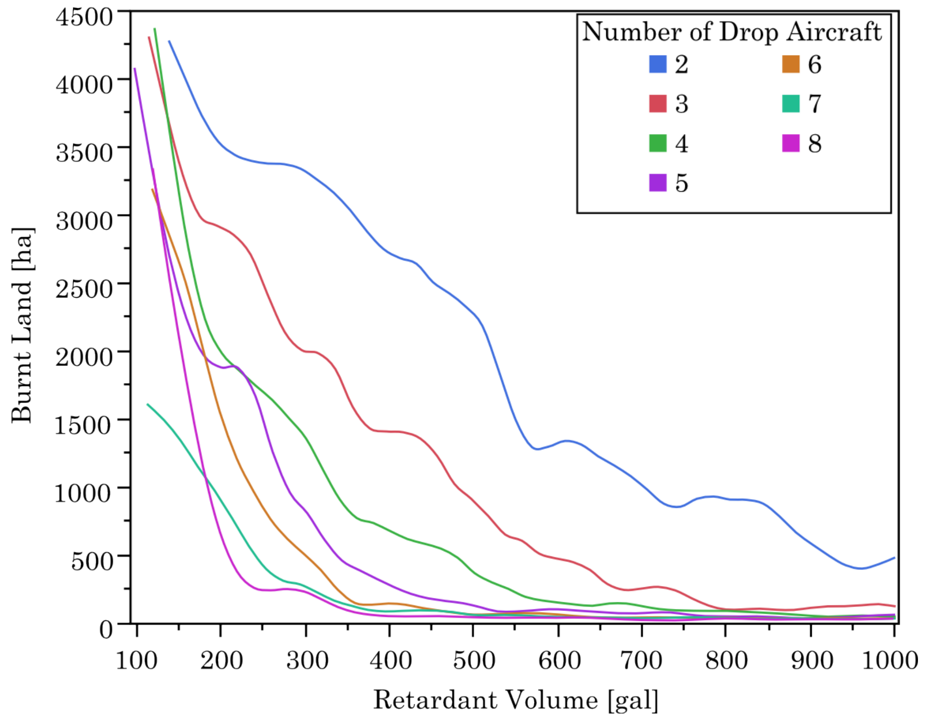


Figure 106: Burnt Land and Drop Aircraft Capability for Varying Numbers of Drop Aircraft

One important trade-off is between the numbers of aircraft and their capability and can be explored by using this SoS model. For instance, figure 105 indicates that the customer may be able to operate fewer search aircraft if a better radar is used or fewer drop aircraft if they each can carry more retardant. The burnt land decreases as payload retardant increases for different numbers of drop aircraft. Because the SoS agent based model is stochastic, there can be significant variation in the output metrics given the same input design variables. Even through the stochasticity, the figure does reveal that for the same number of aircraft, larger more capable aircraft prevent more fire damage. Interestingly if the required amount of burnt land is less than 250 hectares then for 5 drop aircraft it takes at least 400 gals of retardant or 500 gals if only 4 drop aircraft are used. In either case, the total carrying retardant capacity is around 2,000 gals and goes to illustrate the consistency of the firefighting physics.

The number of search aircraft is another important dimension of the problem, figure 107 shows the general trades between numbers of aircraft, burnt land, payload volume, and the fire detection time. This is a complex but useful plot and shows how the burnt land is affected by detection time for

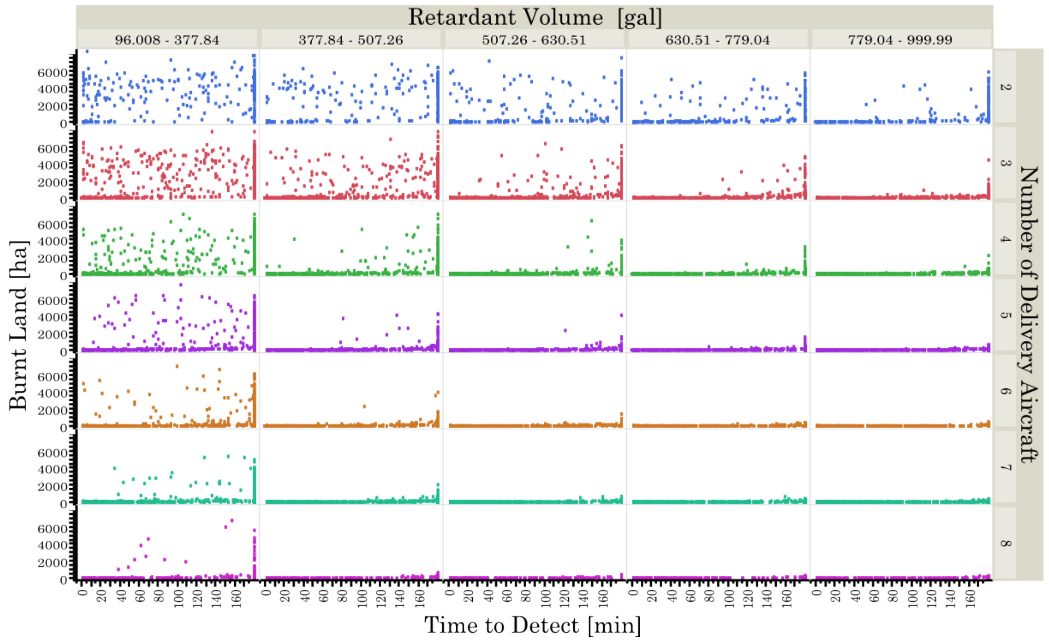


Figure 107: Burnt Land and Drop Aircraft Capability for Varying Numbers of Aircraft

all of the combinations of aircraft retardant volume and the number of drop aircraft. For instance, the top left scatterplot shows the burnt land results vs detection time when using only two drop aircraft with payload capacity between roughly 100-380 gal. The figure shows for fewer smaller aircraft the fire is never successfully extinguished while many large aircraft will always extinguish the fire. The important implication is that there is a trade-off between the numbers of drop aircraft and their size.

The follow trends have been verified: higher number of surveillance aircraft decrease the time a fire continues to burn unnoticed, higher numbers of delivery aircraft reduce the burnt area, and higher payload also reduces the burnt area. The other main objective value for the firefighting scenario is cost. Figure 108 shows the burnt land verses cost envelopes for different numbers of drop aircraft. The envelopes are used because there are many dimensions that affect both cost and burnt land, and at this conceptual stage, the need is to verify the aggregate trend. From the figure, diminishing returns can be seen that the next increment of drop aircraft continues to raise costs while yielding progressively less improvement in extinguishing capability.

Now that the modeling and simulation environment has been verified for the firefighting scenario, the maritime scenario modeling and simulation environment needs to be verified. Again the

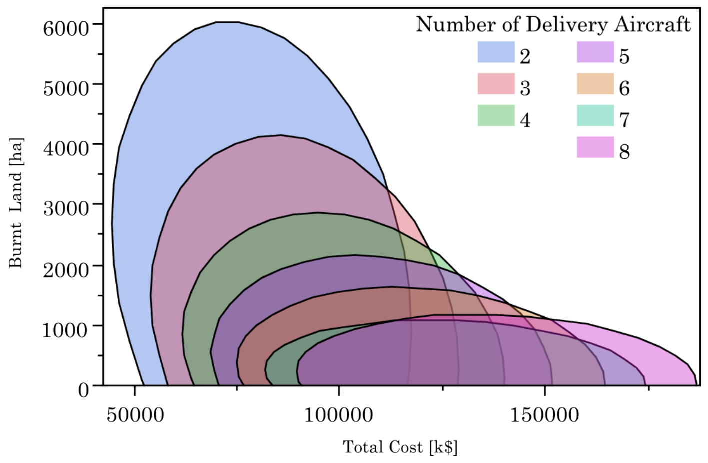


Figure 108: Burnt Land vs Total Firefighting Scenario Cost

objective values found previously from the customer needs for the maritime missions are the cost, the total search and rescue time (SAR), the time to detect oil spills, and the time between two detections. The time between two detections is a metric showing the general ship tracking ability by recording the time ships are unobserved. Figure 109 shows the envelope time it takes for the search and rescue mission for the maritime scenario vs cost for different number of drop aircraft. There again are diminishing returns where the number of delivery aircraft quickly stop yielding an improvement in the total SAR time.

Figure 110 shows the envelop of the time between two detections vs cost for various numbers of search aircraft. The figure indicates using only one search aircraft can not adequately cover the area necessary. As there are more aircraft searching the average time a ship goes undetected drops.

The last trend to verify for the maritime modeling and simulation environment is the time to detect oil spill, figure 111. The figure shows the envelop of spill detection time versus program cost by number of search aircraft. Again as expected, more search aircraft improves the spill detection but eventually increasing the number of search aircraft does not give a significant performance improvement.

In addition to verifying the impact of the number of aircraft it is importance to understand the modeled trends of the radar on mission performance, figure 112. The left half of the figure shows the improvement of the time between two detections for improved aircraft radar range for different numbers of surveillance aircraft. As expected increasing the radar performance does improve the

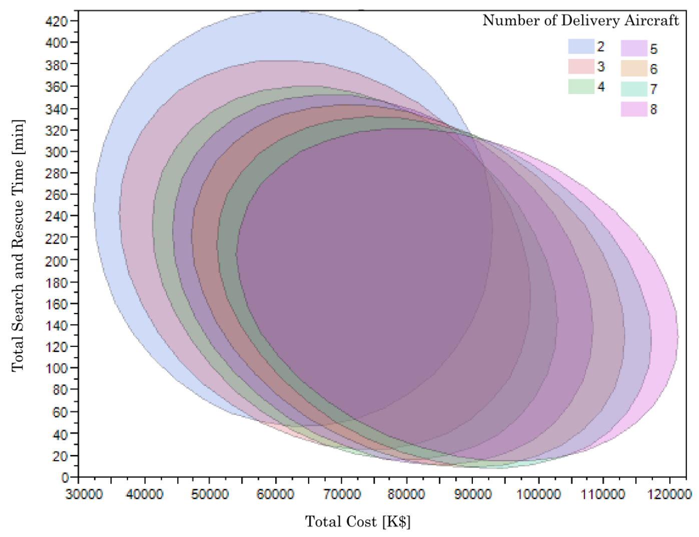


Figure 109: Total Search and Rescue Time Envelope vs Program Cost

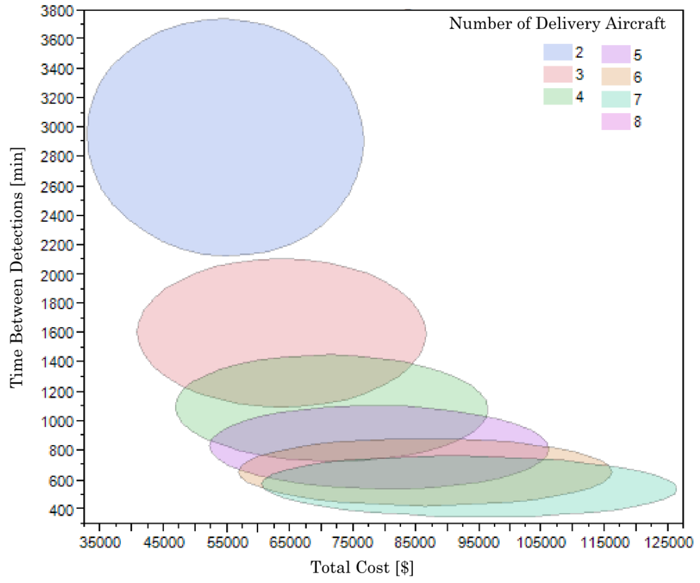


Figure 110: Time Between Two Detections Envelope vs Program Cost

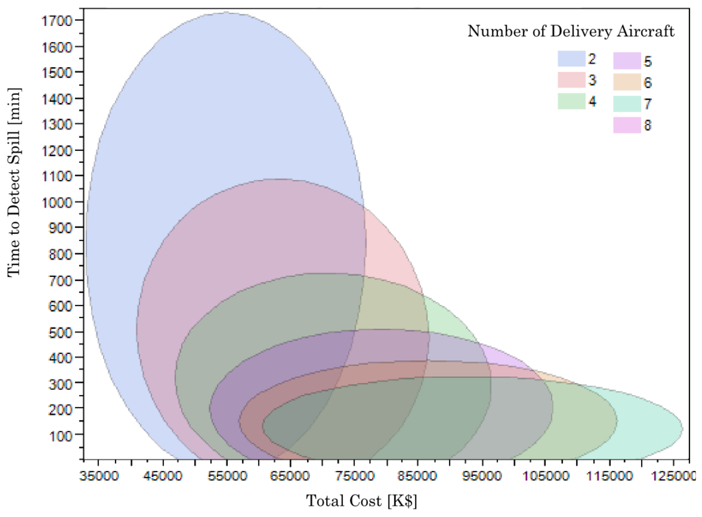


Figure 111: Total Time to Detect Oil Spill Envelope vs Program Cost

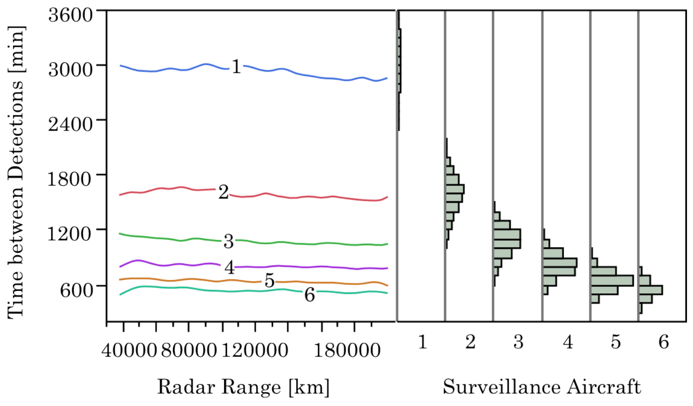


Figure 112: Time Between two Detections Radar Performance

detection capability however the magnitude is surprisingly lower than increasing the number of search aircraft. The right half of the figure shows the distribution of detection time for different numbers of search aircraft. The interesting trend here is that not only does the detection improve the distribution is also significantly tighter.

Figure 113 shows the probability to identify the polluter. This trend is more difficult to capture because the exploration of the design space did not include repetitions. This metric was derived instead by discretizing the design variables and looking at the relative frequency of times the polluters were identified. Overall the trend is as expected, higher numbers of search aircraft with longer range sensors have a better change of successfully collecting the evidence needed to prosecute the

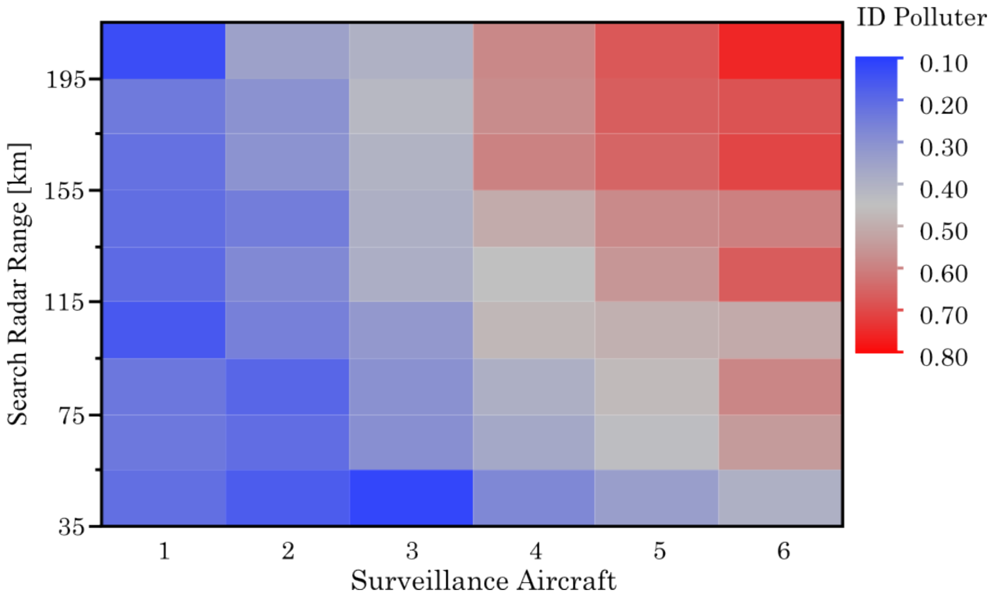


Figure 113: Probability to Identify Polluter

polluter. Because of the processing to generate this probability and the low number of successful database cases, the trend is not well converged. For instance, with one search aircraft a long range radar has a slightly worse probability than short range and reinforces the need for additional cases in the design space exploration.

The last objective value for the maritime missions is the time to cleanup the oil spill. Figure 114 shows the time to perform the search and rescue vs the time to clean the oil spill for different combinations of search and drop aircraft. First low numbers of either times of aircraft yield very poor performance for the missions, and high numbers of both have the best performance. However after 3 or 4 search aircraft and 3 or 4 drop aircraft the improvement is negligible.

This section processed the design space exploration data through different visualizations to help verify the trends in the modeling and simulation environments reflect reality. Overall there is a clear indication that this problem could benefit a design exploration method that focused on the interesting trade-off regions between the numbers of aircraft. Continuing to explore the corners of the design variable ranges would continue to yield points that are incapable of achieving the customer needs either by large program costs or poor mission performances. Running many dominated design cases only serves to waste computational effort without adding actionable knowledge. A denser sampling along the Pareto frontier improves the changes of finding meaningful commonality opportunities

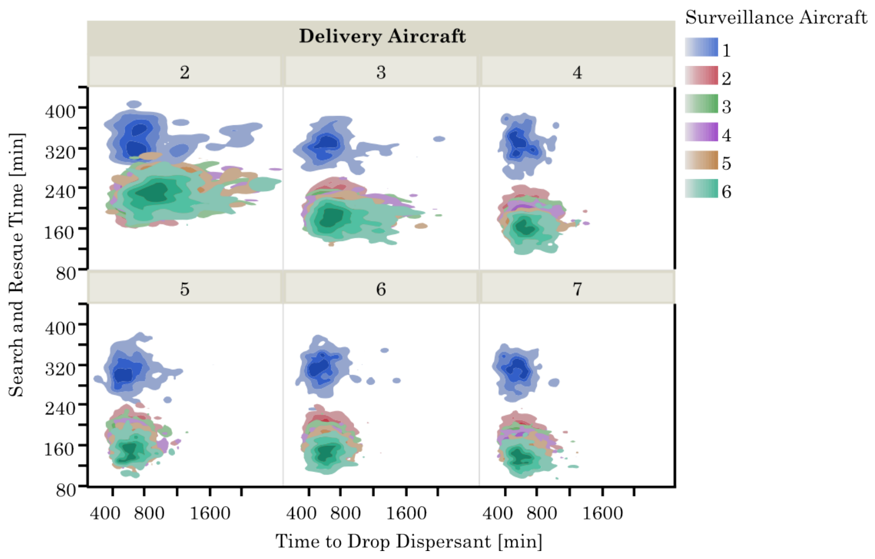


Figure 114: Search and Rescue Time vs Time to Clean Oil Spill for Different numbers of Aircraft

for aerospace product families.

8.5 Commonality Identification

Now that alternatives have been generated and that the trends of the data has been verified, the commonality identification approaches can be tested for the aircraft family.

Without a concrete set of preferences to determine the acceptable trade-offs between metrics, it is not possible to arrive at a final family. However, the purpose at this development stage is to test the commonality identification approaches formulated in this dissertation. In order to make meaningful observations about the approaches, two different filtering criteria are used which represent a reasonable set of alternative preferences that yield a very clear commonality change for the aircraft fuselage. These demonstration filtering criteria are shown in table 24.

The first filtering criteria sets the objective values in an attempt to make the two search aircraft similar and the two drop aircraft similar. It does this by finding the subset with the lowest cost and best mission performances. The second filtering criteria attempts to the make the maritime drop aircraft smaller and more alike to the two search aircraft while making the firefighting tanker larger. This is done by relaxing the oil dispersant mission and time between detections while relaxing the cost requirements for the firefighting scenario. By relaxing the costs, the variability of the burnt land also decreases because design cases with high numbers of large drop aircraft are now

Table 24: Criteria for Filtering

	Criteria 1	**Criteria 2**
Maritime Scenario	Time Between Detections < 80min Total SAR Time < 120min Time to Clean < 430min Total Cost < 100,000k$	Time Between Detections < 700min Total SAR Time < 120min Time to Clean > 900min ID Polluter > 50% Total Cost < 80,000k$
Firefighting Scenario	Burnt Land < 10 ha Total Cost < 100,000k$	Burnt Land < 10 ha Total Cost > 130,000k$

Table 25: Filtered Subset Data Size from Original DoE

	Firefighting	Maritime
Criteria 1	174	77
Criteria 2	481	57

feasible. By comparing these two criteria, it is possible to test whether the commonality approaches developed in this dissertation can scale to a realistic product family design problem. For example, if the commonality approaches can recover this expected change in the family fuselage geometry similarity then it supports their application to refine the combinatorial space in the generic product family design methodology.

8.5.1 Fuzzy Clustering Approach

After applying the filtering criteria, there were only a small set of feasible points remaining from the database, table 25. Figure 115 shows the module space for the resulting aircraft subsets. The left graph is for filtering criteria one and the right is for criteria two. The change in fuselage geometry using the different filtering criteria is clear. For criteria two, the density contours for the maritime drop aircraft are much more similar to the two search aircraft fuselage contours. The fuzzy clustering approach, when processing the clustering similarity, should produce dendrograms that also reflect the expected change in fuselage similarity.

Following the clustering approach as in figure 28 from chapter 4, the respective module subspaces were extracted and clustered. After the membership functions were processed, the component similarity dendrograms are created. Figure 116 shows the dendrogram for the similarity hierarchy of the module partitions for filtering criteria one. As expected, the fuselage for the two drop aircraft and the two search aircraft are paired appropriately. Additionally the engine for the air

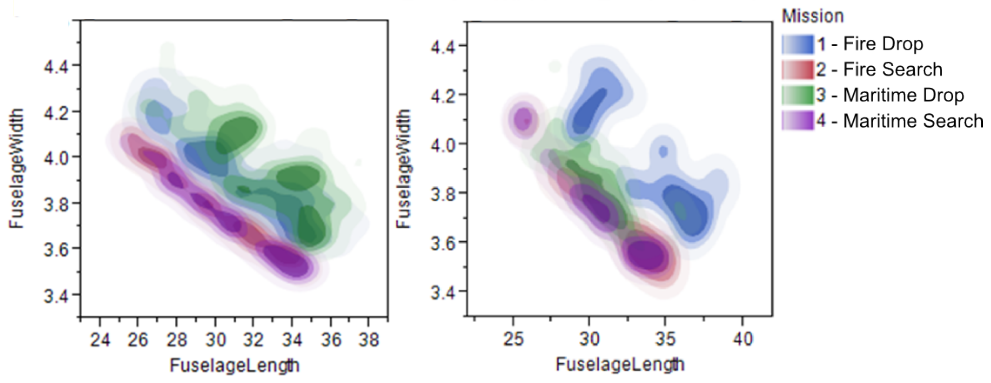


Figure 115: Fuselage Module Space. Criteria 1 (left) and Criteria 2 (right)

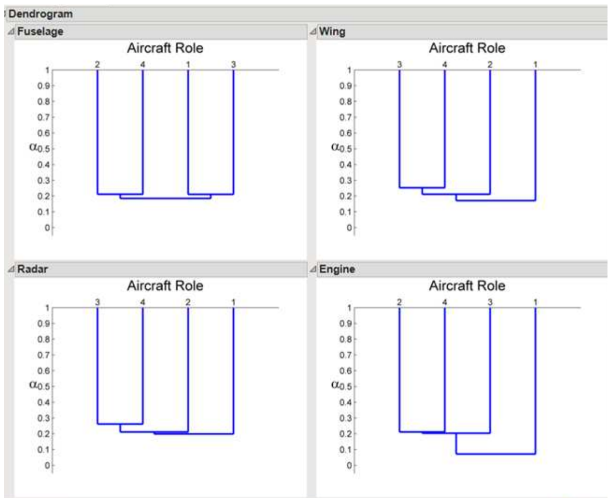


Figure 116: Aircraft Module Similarity

tanker is most unlike the others because it is also the heaviest of the aircraft. The radar can be partitioned between the aircraft for the different scenarios. The difficulty with using these dendrograms is the y-axis scale loses physical meaning with all of the processing necessary to convert the fuzzy clustering membership functions into an equivalence hierarchy.

Figure 117 shows the module similarity dendrograms for the second filtering criteria. Focusing only on the expected change for the fuselage, the fire drop aircraft should be most dissimilar however the dendrogram does not clearly reflect this. This does not invalidate the clustering approach but indicates the need to include more design cases in the database. It is possible a denser sampling would yield a more realistic commonality result. In comparison, the electric motor database held 1

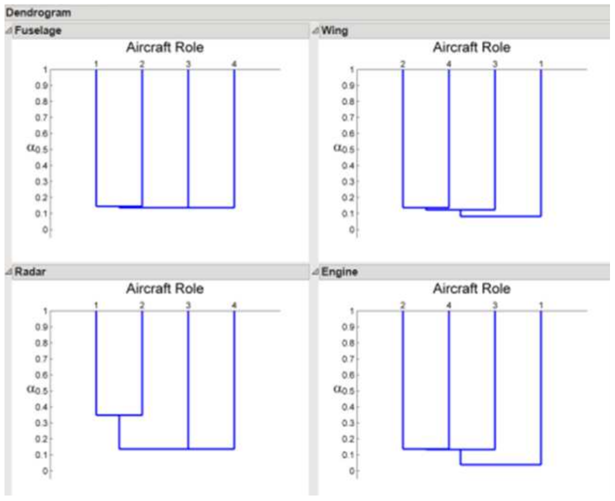


Figure 117: Aircraft Module Similarity

million points and there were only one hundred eighty thousand DoE points to explore the aircraft family which also has significantly more noise.

8.5.2 Commonality Probabilistic Approach

The probabilistic approach also uses the same design space exploration database. To create the probabilistic model, first the structure needs to be learned. The structure learning uses a greedy search that continues to evolve the network so that the result has a better BIC score. The BIC score is based on the likelihood of the model representing the data but penalized for having to train additional connections. This helps ensure only the important variable interactions are taken into account. Additionally there was human interaction between restarts of the greedy search to help remove connections that were unreasonable. The overall structure training took days to generate but was able to arrive at an appropriate network structure. Figure 118 shows the Bayesian network structure for the firefighting design space exploration and figure 119 shows the structure of the maritime database. The figures have had their nodes colored based on their association with a specific aircraft type.

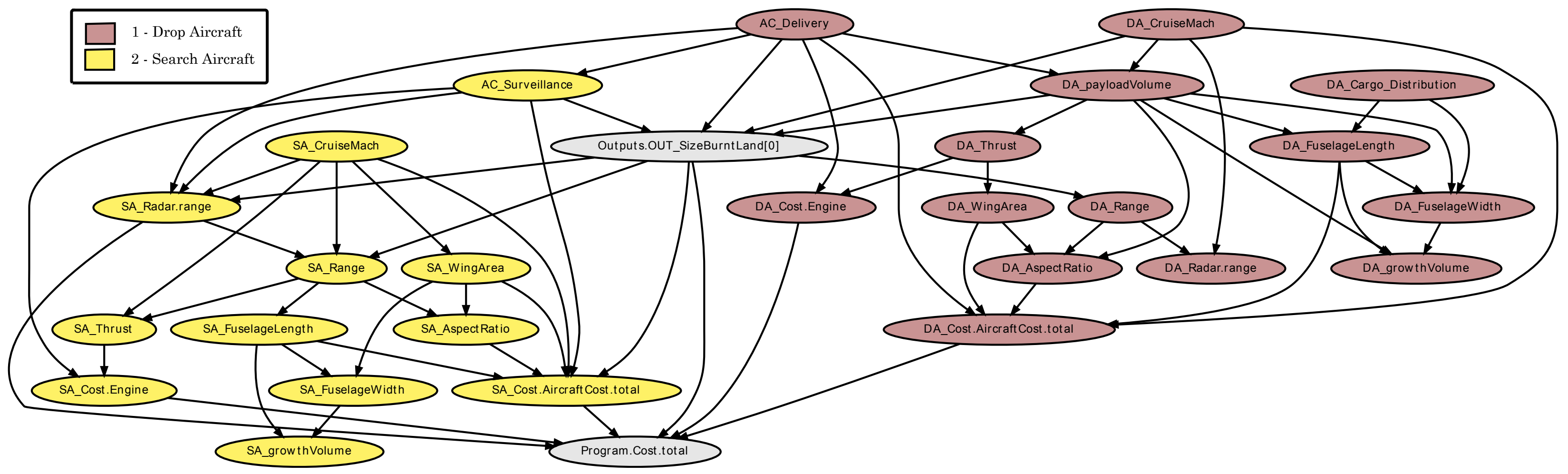


Figure 118: Firefighting Scenario Bayesian Network Structure

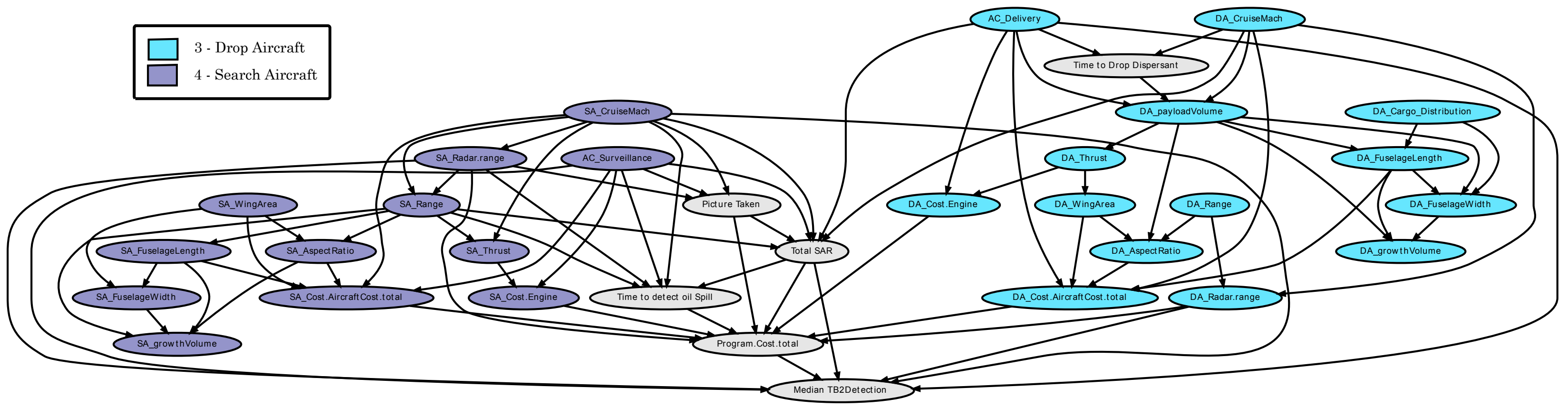


Figure 119: Maritime Scenario Bayesian Network Structure

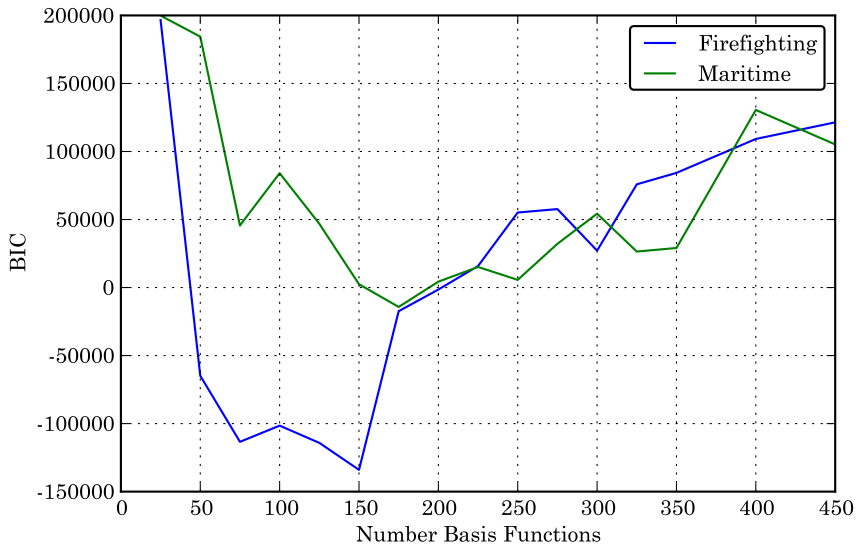


Figure 120: Basis Function BIC Score

Interestingly, the resulting structures show the aircraft nodes are organized by type with the only connections being through the value objective nodes. With this structure, the node conditional densities can be estimated. In addition to the hyperparameters for each of the conditional density estimates, the number of basis functions were varied. Figure 120 shows the BIC score of the networks as the number of basis functions in each node is varied, where low BIC scores are preferred. For small numbers of basis functions, the node conditional densities found have a very low likelihood given the database. As the number of basis functions increase, the conditional densities become more accurate and the overall model likelihood improves. However, as the number of basis functions increase, the penalty for estimating the extra parameters also increases. Eventually this penalty outweighs the benefit from the additional basis functions. The BIC score helps to generalize the network and prevent overfitting. This figure is used to accept the networks trained with 150 basis functions for both the firefighting and maritime scenarios.

Now that the network structures and node conditional densities are estimated, there is a useable Bayesian network surrogate model of the scenario design spaces. Like with traditional surrogate modeling techniques, the next step is to test the Bayesian network model accuracy. Unlike traditional functional surrogate models, the Bayesian network does not have the same statistical goodness-of-fit tests. However, the trained Bayesian network already selected the model with the most likely hyperparameters which were found using a grid search. Additionally, the BIC score helps to prevent overfitting. The remaining model verification involves inspecting the variable interactions encoded by the network. It is accomplished by comparing the training data to a logic sampling from the Bayesian network.

Figure 121 shows the parallel coordinates of the original database along with data sampled from the maritime Bayesian network. The left parallel plot shows the original database for the primary objective variables and types of aircraft. The right parallel plot shows the logic sampling from the trained network. Again if the network has successfully been trained to the data, then it should have encoded the relationships between the different variables and the two plots should appear very similar. Indeed the maritime Bayesian network appears to have captured most of the original distribution with the exception of the "long tails" for search and rescue time and time between two detections. This is probably because the conditional densities fit the main body of points as the likelihood consequence for the tails is smaller.

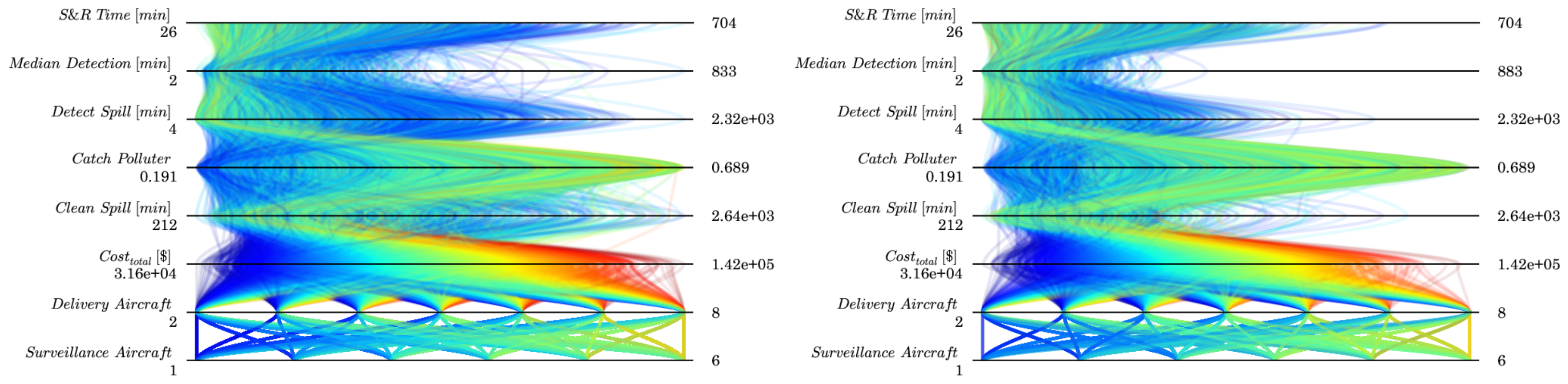


Figure 121: Maritime Scenario Original left, Samples from Bayesian Network right

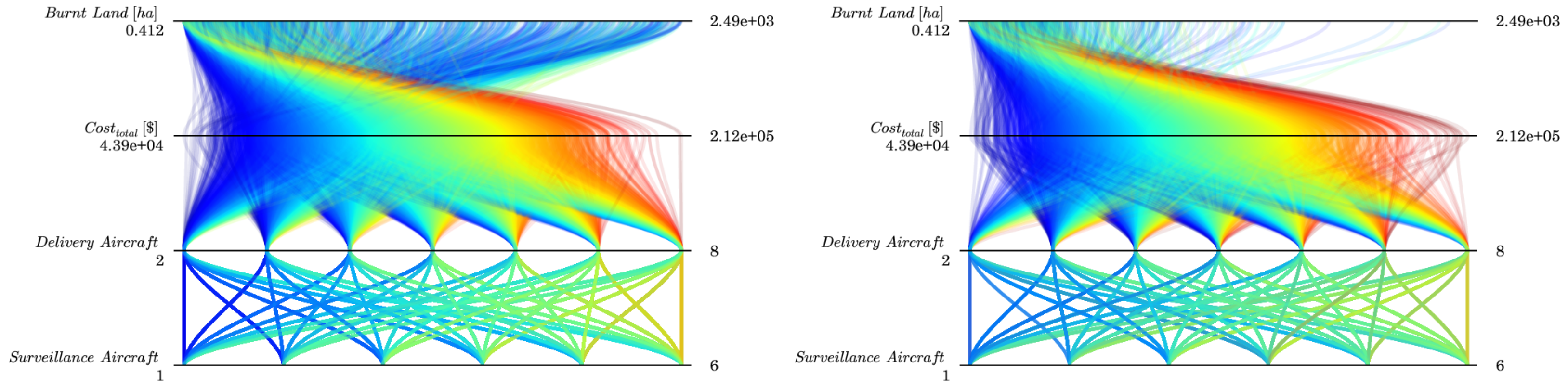


Figure 122: Firefighting Scenario Original left, Samples from Bayesian Network right

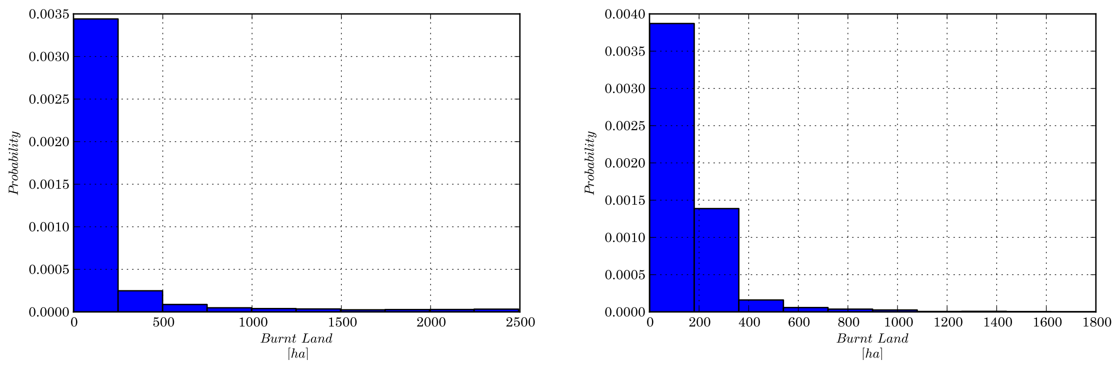


Figure 123: Burnt Land Comparison Original left, Bayesian Network right

In a similar fashion the firefighting Bayesian network can be verified, figure 122. The left shows the original data from the firefighting design space exploration and the right shows the logic sampling from the trained network. The structure of the data also appears to be very close to the original data except for the “long tail” of the burnt land.

This comparison of the burnt land distribution can be seen in more detail in figure 123. The reason the Bayesian network does not fit the distribution for high values of the burnt land is probably because the conditional densities are training to the main trend and not placing basis functions on the outlying points where much of their prediction would be wasted thereby lowering the likelihood. Overall this current limitation of fitting the long tails happens only with design cases that are undesirable. For instance, design alternatives that have large amounts of burnt land are not going to be considered successful, so having higher prediction errors is not going to effect the usefulness in the low burnt land regime.

8.5.3 Comparison of Criteria

With the network models verified, the different filtering criteria can be used to test the commonality reasoning of the probabilistic approach. Using the first filtering criteria as evidence, an importance sampling was performed. Figure 124 shows the resulting sample of the different scenarios on separate parallel coordinate plots. On the left is the maritime scenario network inference sample and the right is the firefighting network inference sample. Again parallel coordinate plots are useful for the product family designer. An important takeaway from these plots is that there are still vary many combinations of design variables that can achieve the criteria one customer needs. To finalize a family, additional constraints are needed to prune more options which will be reflected in a smaller bandwidth on the parallel coordinates.

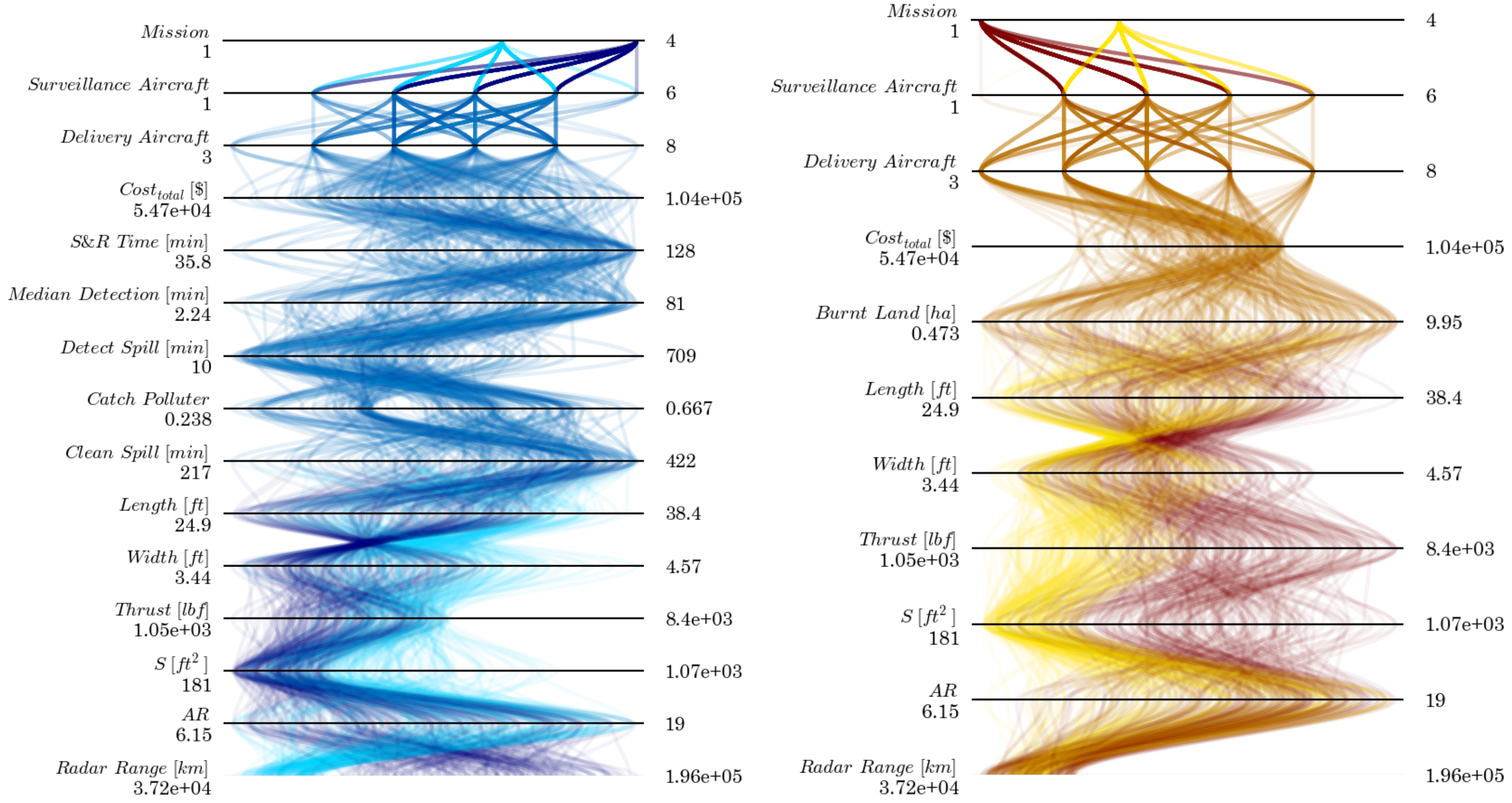


Figure 124: Parallel Plot of Samples for First Filtering Criteria

Similar to the electric motor, the relevant design variables representing the module spaces can be inspected. Figure 125 focuses on the posterior module distributions found from the importance sampling inference. From the filtering criteria there are still rather wide ranges of design variables that would be able to satisfy the requirements and agrees with the large bandwidths in the parallel coordinate plots.

However, similarity between these module posterior distributions can calculated using the EMD to reveal potential platform configurations to evaluate, figure 126. As expected, the fuselage component can be divided between the drop aircraft $\{1,3\}$ and the search aircraft $\{2,4\}$. The weighted graphs also show the relative similarities between the aircraft in the other module spaces. For instance, the radar of the maritime search is very unique which can also be confirmed by inspecting the parallel coordinate plots.

Now that the first filtering criteria has yielded a baseline component commonality, the commonality reasoning inference will be repeated using the second filtering criteria. Figure 127 shows the resulting scenario importance sampling on the parallel coordinate plots. The left is the maritime scenario and the right is the firefighting scenario. While parallel coordinate plots are useful, they do have some limitations. For example, it can be difficult to recognize the change in the maritime drop fuselage geometry.

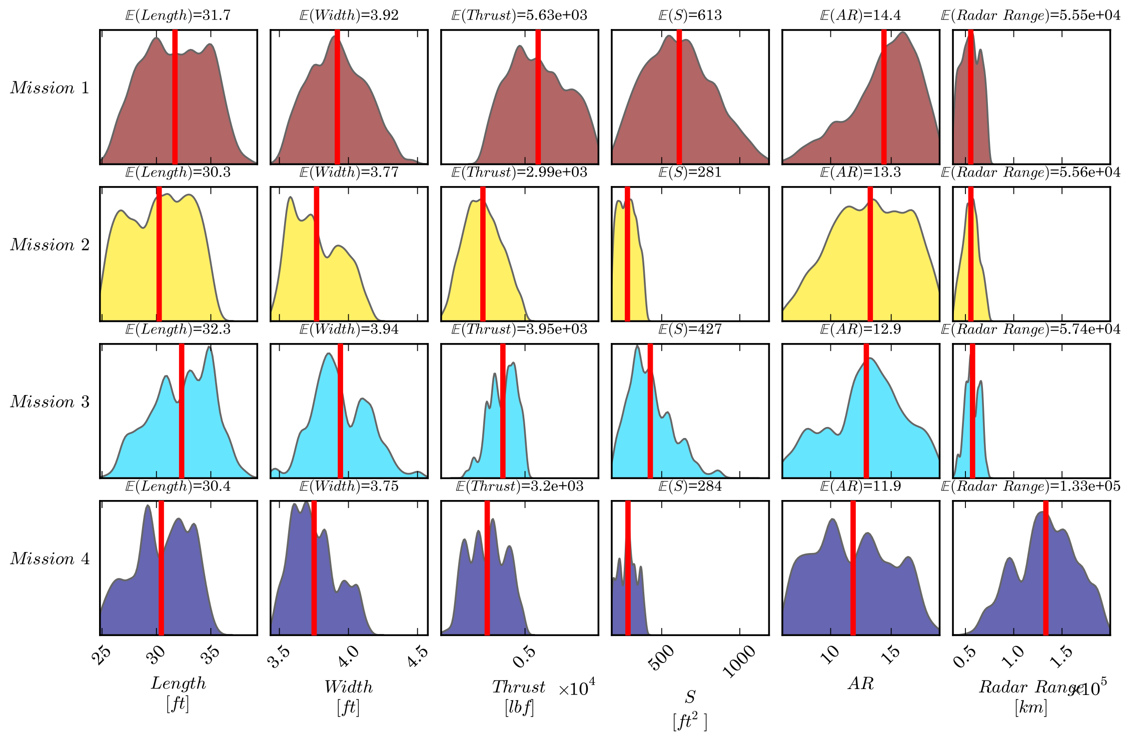


Figure 125: Module Posterior for First Filtering Criteria

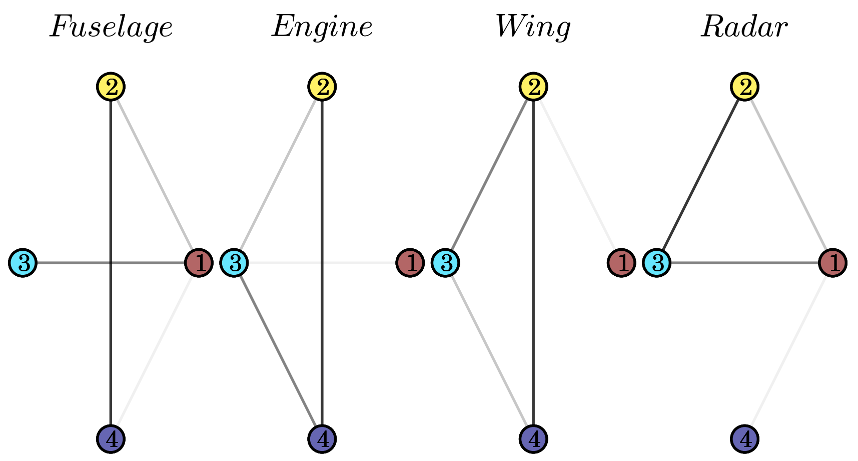


Figure 126: Module Similarity for First Filtering Criteria

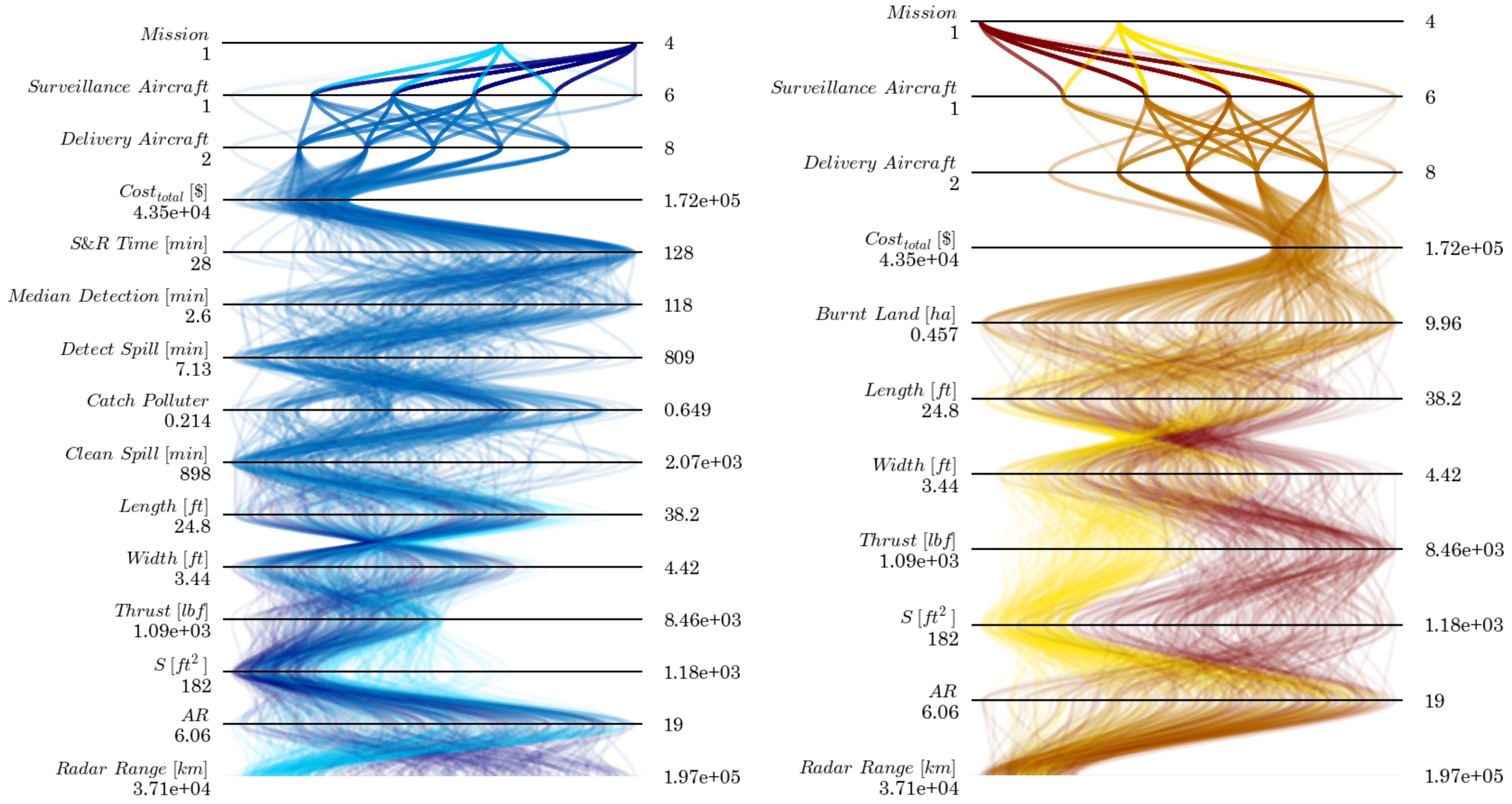


Figure 127: Parallel Plot of Inference Samples for Second Filtering Criteria

The fuselage geometry change is also difficult to see with close inspection of the module posterior distributions, figure 128. There is some indication that the drop aircraft for the maritime scenario (mission 3) seems to appear more similar to the search aircraft for both scenarios but because there are two dimensions to defining the fuselage module space it is not obvious.

However, using the EMD to calculate the similarities between the module posterior distributions reflects the expected change in the fuselage platform configuration. The similarity weighted graph shows the stronger connections between the maritime drop and the two search aircraft {2,3,4}, figure 129.

8.6 Evaluate Alternatives and Make Decision

In the beginning there was no knowledge about which components would make a successful family. The steps up to this point have been performed to enable the designer to make informed commonality decisions that simplify this complex SoS product family problem. This section will describe some of the other capabilities Bayesian network inference can provide to help guide additional exploration aircraft problem. However, if a final family were selected, a detailed set of preferences would need to be known.

Now with the component commonality knowledge extracted with the probabilistic approach, the design alternatives can be evaluated. The probabilistic approach can be a valuable tool in guiding the selection of a final family. Returning to the first filtering criteria from the previous section, importance sampling yields more information about the conditional posterior distributions than just the module distributions.

The posterior distribution of aircraft that satisfy the criteria are shown in, figure 130. This information can be useful for helping to narrow the design variable ranges for additional exploration. Also to help narrow the design variable bandwidths, one additional constraint could be to narrow the number of aircraft operated in the scenarios. By iteratively and interactively using the posterior distributions and changing the inference evidence, the designer can quickly formulate family alternatives.

Inspecting the component commonality weighted graphs, in figure 126 the designer can quickly make some commonality decisions and then combine the relevant module distributions to get a

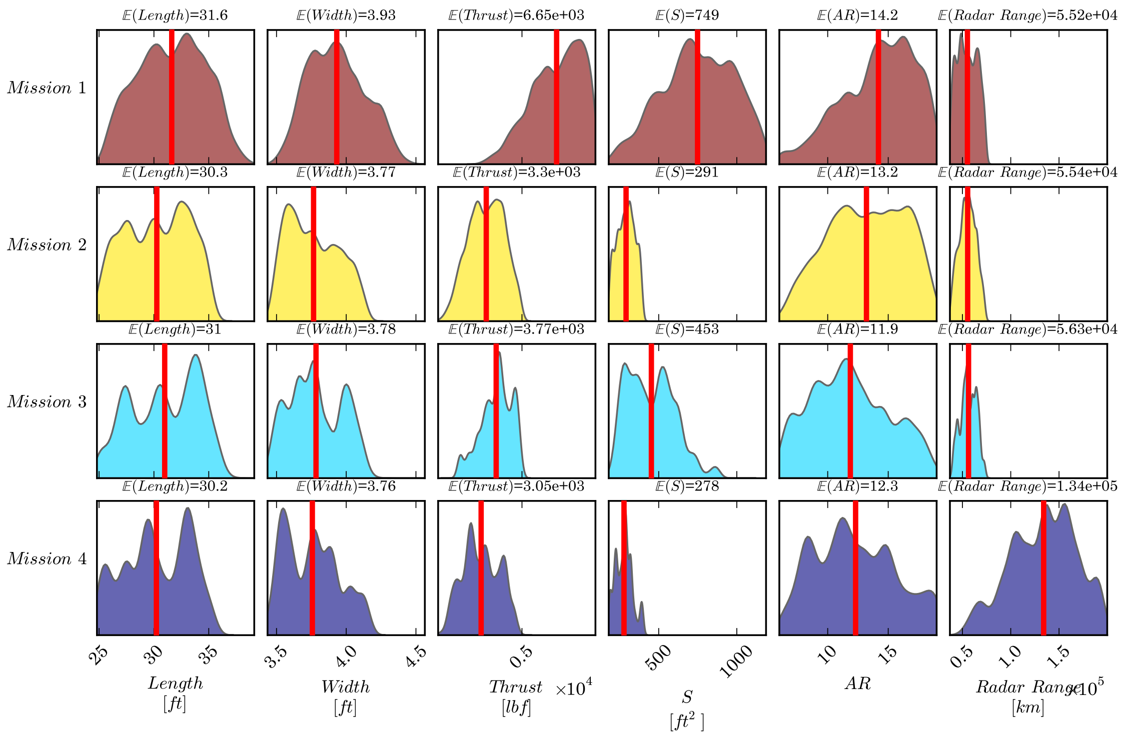


Figure 128: Module Posterior For Second Filtering Criteria

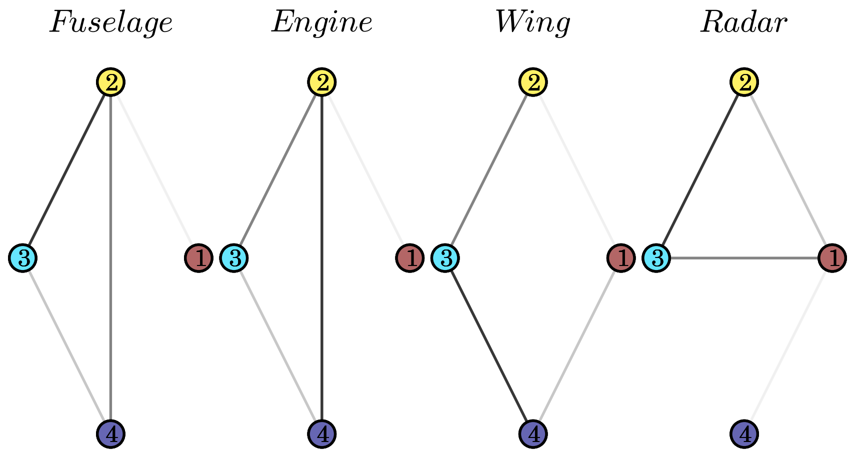


Figure 129: Module Similarity for Second Filtering Criteria

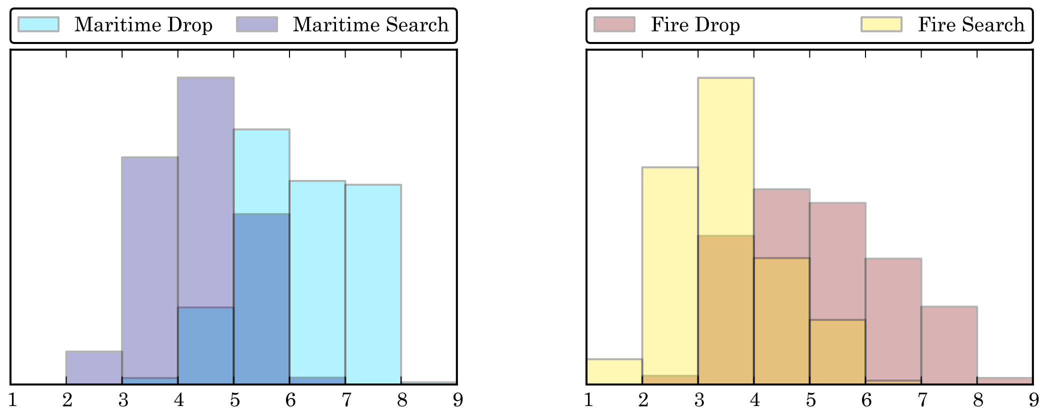


Figure 130: Aircraft Type Distributions

distribution for the platform variables. For example, assuming the engine could be shared between all of the aircraft except the fire drop (1), all of the radars are the same except for the one on the maritime search aircraft (4), and the aircraft fuselage and wings are shared between the drops aircraft and between the search aircraft. Using this configuration the posterior platform distributions can be found, figure 131.

The importance sampling also captures the posterior distributions for the performances given the performance constraints, figure 132. This information could be used iteratively to help guide the designer into alternative regions of the design space. It is also interesting to inspect how the constraints can interact with each other. The main capability of the Bayesian networks is the flexibility of the inference evidence. If this were the final family, the platform variable distributions could be narrowed and used as the evidence for an inference. The inference would then yield distributions of the aircraft family MoEs conditioned on those design variable settings. Alternatively, different combinations of MoE requirements or design variables can be used as evidence for the Bayesian network inference. In the final selection of the family, the MoEs using the narrowed platform design variables would have to be validated using higher fidelity analysis models.

8.7 Conclusions from Aircraft Family Demonstration

The aircraft family design problem is a difficult Class III product family problem mixed with a complex system-of-systems problem. Because there was no knowledge about the ideal family at the start of the problem, it was necessary to formulate the generic product family design methodology.

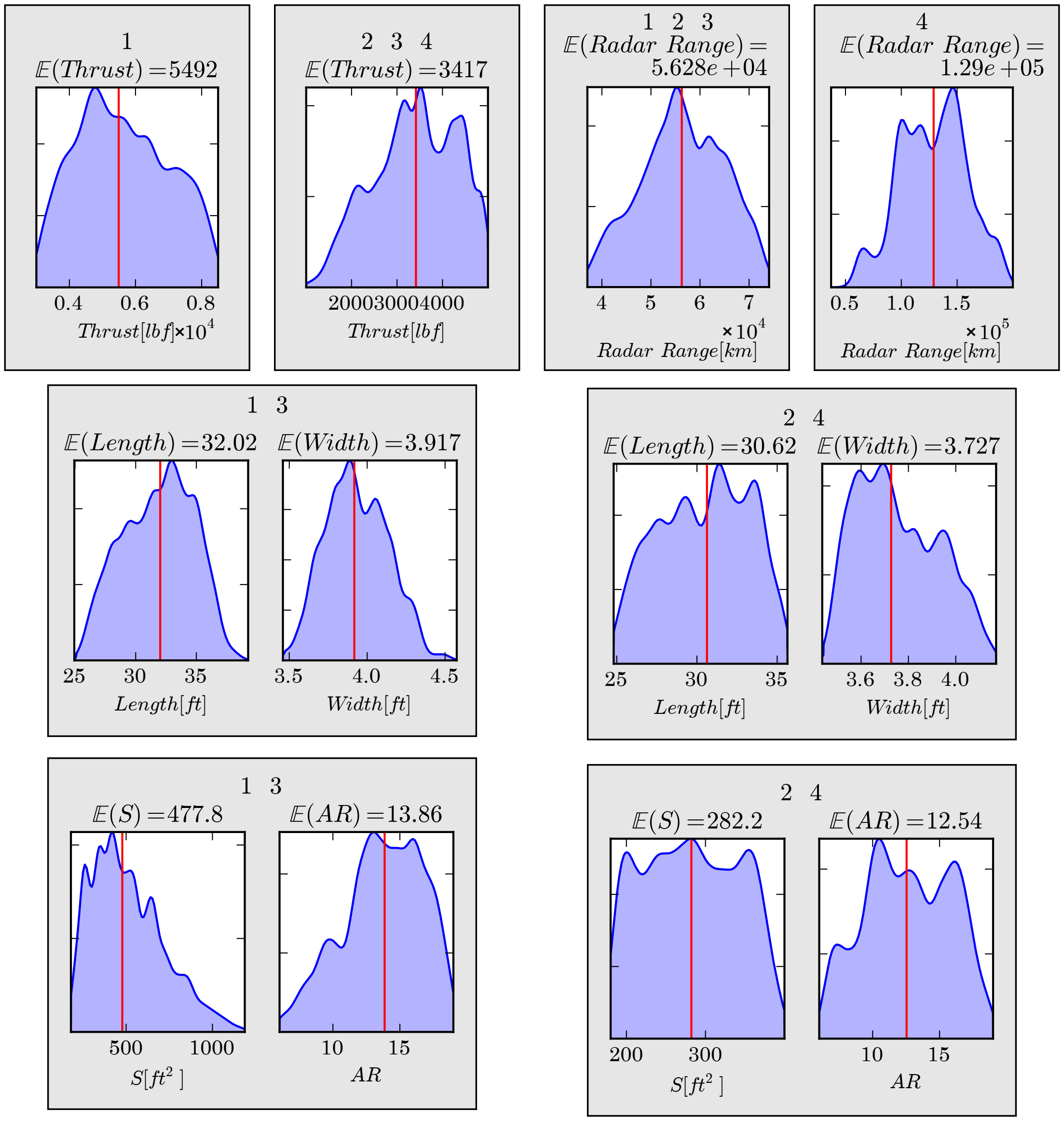


Figure 131: Posterior Platform Distributions for First Filtering Criteria

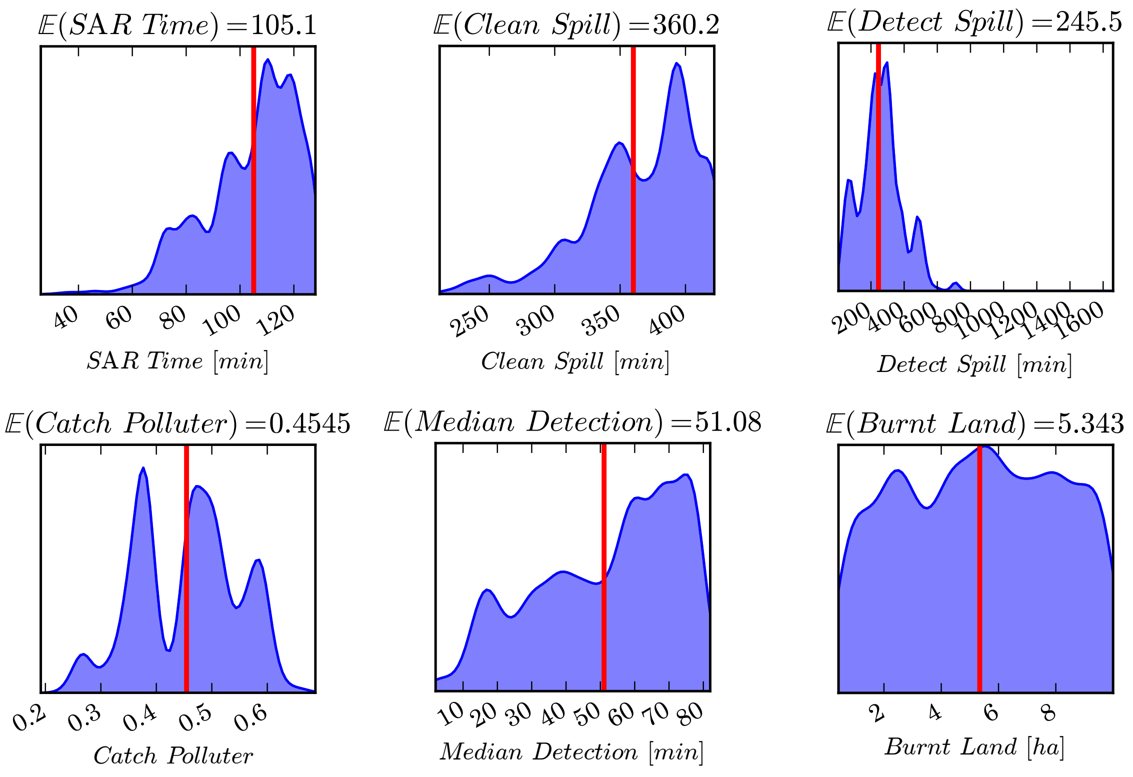


Figure 132: Posterior Distributions of the Scenario Mission Performances using First Filtering Criteria

This methodology is grounded in the framework of the generic IPPD methodology and provides a comprehensive framework for family design problems. The key issue in product family design is the unknown component sharing that drives both the family benefits and product degradation.

This chapter focus on using the aircraft family demonstration problem to help provide context to the generic product family design methodology as well as testing the two formulated commonality identification approaches developed in this dissertation. These commonality identification approaches are necessary because they provide a way for the design to make informed decisions about the family platform configuration.

The aircraft family problem is defined with customer needs at the SoS level where the aircraft are required to operate in two complex and difference scenarios. To begin to understand these scenarios, a detailed analysis of the customer needs was performed. This functional breakdown then identified several possible architectures to formulate the family. The choice of family architecture has far reaching impacts into the final family and component similarities. However, to evaluate the different architectures additional models would need to be created for each alternative. To move forward with the demonstration of the family methodology, a family architecture was selected that involved four aircraft types: one search and one drop aircraft for the fire scenario, and one search and one drop for the maritime scenario.

To understand the implication of the aircraft components on the scenario MoEs, several models had to be created and integrated into a larger modeling and simulation environment. These analysis models range from the component level sensor models like the radar analysis, through integrated vehicle performance code, to the full agent base simulation to capture the SoS interactions. The design space was explored using a LHC design of experiments to generate a database of design alternatives from this modeling and simulation environment. The resulting database and trends from the modeling and simulation environment were verified by identifying several intuitive relationships between the input design variables and the output value objectives.

To test the commonality approaches, two different filtering criteria were used with an obvious change in the fuselage component similarity. Using that database, the clustering approach was performed to find, try, and recover this expected similarity change. Unfortunately, the commonality dendrograms resulting from the clustering approach did not reflect the fuselage change. This is in part due to the sparsity of the design database. The modeling and simulation environment is complex and slow to execute, so the only way to effectively increase the density of the design database it to narrow the design variable ranges.

The probabilistic approach was also performed using the same generated LHC design space exploration database. Two Bayesian networks were trained, one for each scenario. These networks were trained using a greedy search with random restarts and human interaction to remove known poor network edges. Once the network structures were learned, the conditional density of the nodes were estimated using the LSCDE with anisotropic Gaussian kernel basis functions. Using the BIC score, the number of basis functions were varied to select the appropriate number of basis functions needed while limiting over fitting. Finally, the networks were verified by using parallel coordinate plots to compare the training data set variable interactions to a sample generated using the Bayesian network. There were a few inconsistencies with the tails of the distributions but the significant trends were the same.

The two filtering criteria were used as evidence for the importance sampling to generate posterior module distributions. The similarities between two posterior module distributions were calculated using the EMD and allowed for another weighted graph representation similar to the fuzzy clusterings similarity relationships. However, unlike the clustering approach, the probabilistic approach is able to successfully extract the known change in the fuselage similarity between the two filtering criteria. This helps to support the Bayesian network probabilistic approach for use in the generic product family design methodology. In the future, a designer can use the trained Bayesian networks to quickly generate alternative family designs using statistical inference with alternative input evidence.

Chapter IX

SUMMARY AND CONCLUDING REMARKS

In order to capture the largest market share, manufacturers need to offer products to satisfy the widest range of customer needs possible. The traditional approach then is to develop these products independently. While this method may allow manufacturers to produce a variety of products to satisfy a wide range of needs, it is also not without significant disadvantages. For example, product designers for different, but similar, products may duplicate their efforts. A potentially better approach is one that capitalizes on the inter-product similarities by creating modular products capable of sharing common components. Leveraging commonalities between the products reduces this duplicated effort and can streamline design, manufacturing, and maintenance for the whole set of products. However, making components common or a family platform, fundamentally requires a trade-off between the overall family cost benefit and individual product performances. This trade-off becomes worse when combining components together that are dissimilar. This platform configuration problem is difficult because it is a large combinatorial space with many possible options. The platform configuration selection is influential on the overall performance of the product family.

Existing product family design literature divides product family problems into three classes. Class I problems involve a given platform configuration, and seek to find design variable values. These problems are typically the easiest. This class is frequently encountered when there is already significant knowledge about the product family. Class II problems occur when there is a library of components that the product family must be built around. In Class III problems both the component design variables and platform configuration are unknown. This class is the most difficult because it incorporates the large combinatorial platform configuration space. While there are many existing design methods, the literature shows there is a need for additional methods for addressing complex combinatorial Class III product families.

Moreover, none of these classes of problems account for interactions between products at the system-of-systems (SoS) level. This is a significant drawback of this class scheme. Such problems are even more complex, because now the product requirements are unknown along with the design variables and platform configuration. This adds an additional layer of complexity when the individual product requirements can be traded against each other while maintaining the overall customer needs. Some product family options may meet the customer needs but may limit commonality. When approaching a new family design without any prior knowledge, assuming a product family platform configuration may lead to a suboptimal family. Starting from this standpoint of a generic SoS product family design problem with no existing knowledge, it was hypothesized that the key was to lessen the platform configuration combinatorial problem by identifying potential product component similarities, hypothesis 1.

Hypothesis 1

Incorporating knowledge about potential family platforms into a larger product family design methodology can focus computational resources away from considering poor platforms, making the design process more efficient, and helping to identify the ideal trade-off between product performances and portfolio costs.

This is a broad hypothesis and is divided into smaller sub-hypotheses that are more testable. Sub-hypothesis 1.1, focuses on the question for why finding commonality is key for the methodology.

Sub-Hypothesis 1.1

If poor component combinations can be eliminated from consideration, then design resources can focus on identifying the ideal tradeoff between product performances and portfolio costs.

Sub-hypothesis 1.1 is easily tested with a small thought experiment. In a manner similar to a continuous space screening test, pruning known poor options from a discrete space frees computational resources from evaluating unimportant relationships. These resources can then concentrate on better understanding the main critical relationships. The other part of the main hypothesis is broken down into sub-hypothesis 1.2.

Sub-Hypothesis 1.2

If combining components that are more similar makes a better family platform than combining dissimilar components, then a method to extract these patterns will aid in the formulation of the family.

This hypothesis establishes the need to develop an approach for systematically identifying family opportunities by identifying component similarities.

This dissertation proposes such a method based on the Integrated Product and Process Design (IPPD) as a framework. The first step in this method involves identifying the customer needs. In a family problem this may require segmenting the market by dividing similar customer needs into a niche. Using these niche customer needs, a detailed functional analysis captures the different tasks necessary in achieving them. For the case of an SoS of aircraft, this involves defining the relevant missions. The next step in this method, the family architecture must be defined. The architecture allocates function to particular products. Once the family architecture is defined, the importance of the different objective values must be specified as well as the different module space parameters. This along with the family architecture sets the requirements for the creation of a modeling and simulation environment capable of seeing the impacts of low level module design parameters on the high level customer needs.

Next, the design space can be explored to generate a database of feasible alternatives. Then the database can be inspected to identify commonality that will help reduce the combinatorial problem involved with the family platform configuration. Finally, these alternate platform configurations can be explored in additional detail to select the family ideal according to the appropriate preference scheme selected.

In addition, this dissertation also proposes two alternative approaches to identify component commonality. The first approach, formulated in chapter 4, employs clustering to identify component commonality. Fuzzy clustering is a technique that groups more similar points together. In the context of clustering product design alternatives, if components from the feasible database subset are similar, they are grouped into the same clusters. Through a series of fuzzy operations to process these clusters, an overall similarity relationship between product components can be found.

This binary fuzzy equivalent relationship then indicates the similarity two products have towards each other. Knowing that combining these similar components yields a better family platform than combining dissimilar components, this binary fuzzy equivalence relationship can serve as a basis for identifying family platforms to guide future exploration. Sub-hypothesis 2.1 describes the clustering approach's commonality identification ability as predicated on the quality and density of the design space database.

Sub-Hypothesis 2.1

If a sufficiently accurate and dense database can be generated, then a machine learning pattern recognition technique like fuzzy clustering could be used to help identify component commonality and potentially form a platform.

Sub-hypothesis 2.1 is tested using two experiments. Experiment 1 focuses on comparing the clustering approach to a baseline electric motor product family design study. The first part of this experiment qualitatively compares the dendrograms found between the two approaches. This comparison confirms that the equivalence hierarchy does reflect much of the same component similarity relationship structures. The second part of this experiment uses the baseline platform configuration to see how close the feasible subset of the clustering approach compares to the target design variable settings. This experiment confirms that the fuzzy clustering approach to commonality identification is able to recover similarities between the different products' components but is sensitive to the density of the design space exploration.

Experiment 2 further investigates the reliance on the design space exploration method to generate the database to be clustered. Three different exploration methods were used: a uniform Monte Carlo, Latin Hypercube, and a Multiobjective Genetic Algorithm. The number of design cases in the database for each of the exploration methods were varied as well. Both the LHC and MC methods perform the same with high numbers of design cases. This is because the density of points near the baseline target values is roughly the same. When there are very few cases in the design database, the MC method can "get lucky" and draw several points near the targets. The LHC method gives a more consistent improvement of the overall quality of prediction for the clustering approach. When the design ranges were refined, both the MC and LHC explorations performed roughly the same with the LHC slightly better. The multiobjective evolutionary genetic algorithm is included in the design space exploration experiment because this method intelligently generates points from previous generations to improve the quality and density of points towards the Pareto frontier. When using an intelligent exploration method, care needs to be taken that the groups found from the exploration are natural and not artifacts of the exploration method. For this problem, however, the results derived from the database from the multiobjective genetic algorithm yield an improvement to the estimate of the platform variables over the space filling exploration approaches.

Overall this experiment confirms that the clustering approach's platform prediction estimate improves with the quality and density of the design database. Together these experiments confirm that the fuzzy clustering approach is logically sound. However, when the problem is limited by the number of sample points, like with slow complex analysis codes, the clustering approach has wider bounds around the estimated platform variables.

One other important criteria for a product family with long life-cycles is the need for long term flexibility, as noted in observation 1. Sub-hypothesis 3.1 postulates that the fuzzy clustering could be performed with different feasible subsets centered around alternative filtering criteria which represent possible future requirements. These two criteria can then be compared with the resulting extracted commonality relationships to make an informed family platform decision.

Sub-Hypothesis 3.1

If the design constraints are changed, then using the new feasible subset from the design database and performing the pattern recognition will reveal the sensitivity of component sharing.

Sub-hypothesis 3.1 is tested with an additional experiment. Experiment 3 uses a different electric motor design study with a different set of performance requirements and a platform configuration. If the clustering approach can accurately identify the platform variable changes between the studies, then it might be applied to small changes in performance requirements and support sub-hypothesis 3.1. However, the general estimated platform distributions remain wide, which limits the applicability for this approach to only small scale problems that can be densely sampled.

The second commonality identification approach, is formulated in chapter 5. This approach builds a surrogate model of the design space exploration database. So instead filtering the database directly, the surrogate model enables the interpolation between the design alternatives. The kind of design space being explored is a stochastic SoS, which is difficult for traditional functional surrogate models to fit. This dissertation focused on applying a continuous nonparametric Bayesian network to capture the joint probability distribution of the design space exploration database, sub-hypothesis 2.2.

Sub-Hypothesis 2.2

If a model can be generated that encodes the joint probability distribution, then component similarities can be inferred given performance constraints.

After fitting the Bayesian network, inference techniques use the customer needs and other requirements to find posterior component design variable distributions. These posterior distributions describe the likely values of the component design variables for the different products. The similarity between product components can be quantified by calculating a similarity metric, like the earth movers distance, between two posterior component distributions.

Because Bayesian networks are only now being applied to more continuous problems, chapter 6 performs two tests to validate the Bayesian network feasibility as a surrogate model. The first test looks at the effect that the kernel function choice has on the prediction capability. For high correlation between design variables, like what happens on a Pareto frontier, an adaptive anisotropic Gaussian kernel was found to be acceptable. The adaptive anisotropic Gaussian kernel improves the prediction by incorporating local correlations between the design variables. The other test fits a Bayesian network surrogate model to the design space exploration of the Breguet range equation. After training the network, several importance sampling inferences were performed. These tests help validate the implementation and theory behind using Bayesian networks as a surrogate model and helps to support their use in the probabilistic approach to identify component commonality.

Experiment 4 then compares the probabilistic approach to the clustering approach using the electric motor baseline study.

Sub-Hypothesis 3.2

If the design constraints are changed, then any changes to the posterior distributions from the probabilistic model will reveal the sensitivity of component sharing

Finally, the sub-hypothesis 3.2 raises the same requirement for the probabilistic approach to be able to detect family commonality. This is tested in the aircraft family design demonstration problem by comparing the commonality changes between two filtering criteria. The demonstration problem is performed by following the generic product family design methodology that was developed in chapter 3. In the aircraft demonstration problem, two realistic scenarios are used for the aircraft mission set. The first scenario is a maritime monitoring mission whereby the unmanned aircraft need to be able to perform search and rescue, and oil spill monitoring and cleanup. The other scenario is an aerial firefighting scenario that uses unmanned aircraft to detect the fire and drop any dispersants. These are complicated scenarios and a set of models were created to be able to understand the impacts of module level design variables to the high level customer needs. It was found that the clustering approach was not able to detect the expected commonality change but this is probably due to the sparsity of the design space exploration of the aircraft family. However using the same database, the probabilistic approach was able to recover the expected commonality change between the two filtering criteria. This supports sub-hypothesis 3.2. Overall the demonstration problem helps to support hypothesis 1. The fuzzy clustering approach is found to scale poorly to large design problem with wide variable ranges. Table 26 gives and overview of the linking between the research questions, hypotheses, the experiments, and the results.

9.1 Contributions

One of the major gaps in product family design methodologies is the selection of the platform configuration. This dissertation proposes a generic product family design methodology for difficult system-of-systems product family design problems with no prior knowledge. The goal of this methodology is to integrate existing tools with additional methods to identify component similarities to scope the combinatorial problems that arise. This methodology was demonstrated using the design of an aircraft product family design. Significant effort and time were devoted to creating the

Table 26: Summary of Hypothesis Tests

Research Question	Hypothesis	Tested By	Supported?
RQ1 Methodology	1	Sub-Hypotheses	Yes
	1.1	Simple Experiment	Yes
	1.2	H2.1 and H2.2	Yes - through probabilistic approach
RQ2 Commonality	2.1	Ex1 & Ex2	Yes, but can scale poorly
	2.2	Ex4	Yes
RQ3 Flexibility	3.1	Ex3 & D	Partially, limited by database size and quality
	3.2	D	Yes

models necessary to enable the analysis of two scenarios. The resulting analysis environments have already proven a useful capability and been employed in additional research.

This research also develops two possible approaches to fill commonality identification need in the generic product family design methodology. The first approach is developed using fuzzy clustering and based on a theory for using fuzzy operators to extract binary similarities relationships. With these relationships, the hierarchical nature of the component commonalities are graphed and verified against similar approaches.

The other approach commonality identification approach, formulated for this work, is the probabilistic approach. The probabilistic approach introduces a novel continuous nonparametric Bayesian network model to encode the joint probability distribution of the product design space. Posterior distributions of component design variables, conditioned on the product family constraints, can then be extracted using statistical inference importance sampling. Similarity relationships are then generated uses these.

This study also tested the continuous nonparametric Bayesian networks, to support its use as a robust probabilistic surrogate modeling technique. These tests revealed the need to include anisotropic Gaussian kernels in the nonparametric node Least Squares Conditional Density Estimation (LSCDE). Including these anisotropic Gaussian kernels required updating an existing method with a more flexible kernel function. Furthermore, by using the anisotropic Gaussian kernels this method can be applied to represent a manifold, like the Pareto frontier. This then introduces the capability of effectively creating a probabilistic model of the Pareto optimal design region with interesting future applications.

Bayesian networks are computationally demanding, so it was paramount to have an efficient implementation. This thesis work involved heavily extending Pebl, an existing Python library, to be able to perform the training needed for continuous anisotropic LSCDE needed in the probabilistic approach. Additionally the capability was added to perform importance sampling which aids in the inferences necessary to make conclusions with the Bayesian network.

Finally several experiments were performed to understand the platform prediction capabilities of the approaches. As previously stated the fuzzy clustering approach is limited by the ability to explore large design spaces. However the experiments do indicate that the fuzzy clustering approach can find commonality, if the database is dense enough with high enough quality. These experiments also validate the use of the probabilistic approach to represent the design space explorations and that through the use of importance sampling module similarities can be calculated.

9.2 Recommendations and Future Work

The clustering approach is best used when the analysis codes are fast to execute and when the design database has a high quality and near the appropriate design region. If the design space is too high dimensional with wide variable ranges the number of cases needed for an accurate estimate may be prohibitive for complex codes. However, if accurate is not as big of an issue it may be beneficial to continue to use the clustering approach. For example, if the goal is to scope the family design problem down, it may be useful to identify the dissimilar components and to narrow the design variable ranges.

The probabilistic approach requires encoding the joint probability distribution of the design space. This is a difficult problem when the number of dimensions is high. For comparison, the aircraft family had a network size of 30 dimensions. If there is no knowledge about the variable interactions to help guide the structure learning the factored distribution could be suboptimal or miss important variable connections making the inferences less accurate. The LSCDE process is not as difficult but still can be time consuming because it involves a grid search to find the hyperparameters. Also the evaluation of the network of inference can be computationally demanding. For example 100,000 samples may need to be generated to yield an accurate importance sampling. With high correlations between nodes that limit back propagation, or with unlikely evidence it may take a larger importance sample.

To continue to improve on this inference capability, especially in the presence of unlikely evidence, the library could be further expanded in the future to include alternative sampling techniques like MCMC (Gibbs sampling, or Metropolis-Hastings), or message passing sampling techniques. In addition to a more efficient sampling technique, the implementation can be expanded to utilize distributed computing. Many of the Bayesian network processes needed for the training and inference are embarrassingly parallel and distributed computing would yield large speedups. Another improvement for the library could be a more flexible kernel implementation allowing other nonstandard kernels to be used in the LSCDE.

From the product family design standpoint the next step is to use the probabilistic approach interactively to explore the family design alternatives and to verify the final family with additional evaluation cases using the modeling and simulation environment. Any enhancements to the Bayesian network inference capability will translate into improve performance for family studies.

With increasing computing power, Bayesian networks and other probabilistic graphical models become reasonable surrogate modeling techniques. As this research shows, combining the fast and efficient Python implementation, importance sampling inference can be employed to extract the encoded knowledge inside the Bayesian network. However, this method can be extended in the future for other complex Pareto frontier design problems. To the author's knowledge this has not previously been applied to engineering design and could have important implications in design problems well beyond product families and is worthy of further study.

Appendix A

BAYESIAN NETWORK LIBRARY

A.1 Library Overview

For this dissertation, there was extensive code development. Python was chosen as the programming language because of its ability to rapidly prototype as well as its healthy ecosystem of libraries. This ecosystem includes several scientific computing libraries that simplify the needed code development. The Python Environment for Bayesian Learning (Pebl) is an open source library with many implemented classes and methods. Pebl's original purpose was for learning the Bayesian network structure for discrete variables. The base library already had the network object as well as multinomial conditional probability distribution object for calculating the conditional probability table for the discrete nodes. To learn the network structure, Pebl has several alternative learning algorithms implemented, but the Greedy Learner was used for this work.

The original Pebl library did not use continuous variables or have importance sampling inference capabilities. As discussed in Chapter 5 and Chapter 6, the method of Least-Squares Conditional Density Estimation using adaptive anisotropic Gaussian kernels was selected to estimate the conditional densities of the nodes. The LSCDE implemented here was based off of the original Matlab code from Sugiuama[154]. The original implementation uses isotropic Gaussian kernels which helps to simplify several of the necessary integrals. To be able to efficiently estimate the conditional node densities using the anisotropic kernels, several of the computationally intensive functions were implemented using Cython. Cython is a language similar to Python but with the ability to declare variable types which enables a compiler to convert the code down to C. This has the effect of significantly speeding up the computation time. The network training is complete once the structure is learned and the conditional densities are estimated using the LSCDE.

To use the completed network, an inference engine module was developed. This module combines one or more trained networks with a collection of inferences which are intern a collection of observed evidence target nodes. These targets can be either normal distributions or uniform distributions. The engine is responsible for organizing the different networks and calling the associated inference object's sample method. This method then will perform the importance sampling as described in Chapter 5. If the node is part of the observed, then the importance sample is drawn using the evidence target distributions. If the node is not observed, then the sample is drawn using the anisotropic Gaussian mixture model with the truncated normal Cython method. Figure 133 shows a simplified structure of the extended Pebl library to showcase the relevant objects and flow.

Once all a sufficient number of the samples are taken, the engine can compare the posterior probability distributions between the resulting sample objects. Using these posterior pdfs, the engine can plot the marginal distributions and generate a pairwise similarity for the different module spaces. It is this comparison that serves as a basis for the component similarity relations. The entire source code for the extended library is too extensive to include in this document. However, the modified anisotropic Gaussian Kernel module and Cython helper functions are included. The full extended library code is available upon request.

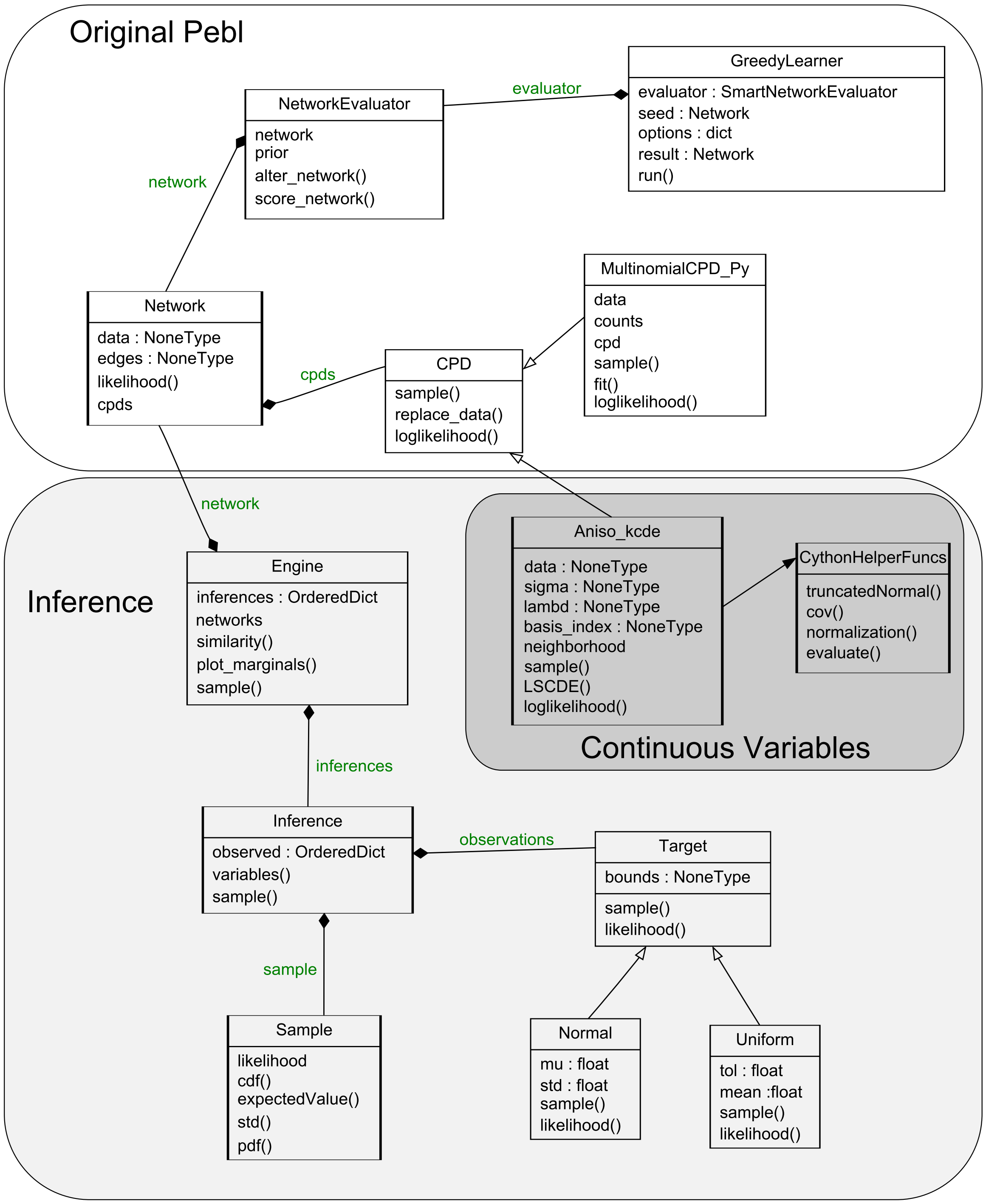


Figure 133: Simplified Library Structure

A.2 Anisotropic Gaussian Kernel

```python
from __future__ import division
import numpy as np
import math
import matplotlib.pyplot as plt
import time
from scipy.spatial.distance import cdist
from scipy.special import erf
from kde import kde
from sklearn.cluster import KMeans
from scipy.interpolate import interp1d
from collections import OrderedDict
import mypebl.distributions.c_kcde
from mypebl.distributions import c_kcde
from mypebl.distributions import dist
from myMDAO.lib.plot.api import scattermatrix
import logging
root = logging.getLogger()
root.setLevel(logging.DEBUG)
root.addHandler(logging.FileHandler("info.log"))


mylogger = logging.getLogger('mylogger')
mylogger.addHandler(logging.StreamHandler())
mylogger.setLevel(logging.ERROR)  # <-- that's the default level, actually
mylogger.setLevel(logging.NOTSET)  # <-- that's the default level, actually


def mahalanobis2(z, w, Vi=None):
    '''Squared Mahalanobis Distance
    Calculate q.T*M*q between point n and basis b

    xa : [dy+dx,n] points
    xb : [dy+dx,b] basiss
    cov : [b][dy+dx,dy+dx] list of covariance matrices
    vi : [b][dy+dx,dy+dx] list of inverse covariance matrices

    return : [b,n]
    '''

    d_z = z.shape[0]
    n_basis = w.shape[1]
    if d_z != w.shape[0]:
        raise Exception('Dimension Error : d_z %s and w.shape[0] %s' % (d_z, w))

    dist2 = dist.mahalanobis2_blas(z.T, w.T, Vi).T

    if np.any(np.isnan(dist2)):
        raise Exception('Error dist NAN')
    return  dist2


def kern(x, y, w, Vi=None):
    '''Anisotropic Kernel'''
    z = np.vstack([x, y])

    d_z = z.shape[0]
    if d_z != w.shape[0]:
        raise Exception('Dimension Error : d_z %s and w.shape[0] %s' % (d_z, w))

    q_dist = mahalanobis2(z, w, Vi=Vi)
    return np.exp(-0.5 * q_dist)


def approx(f, lower, upper, n=100, err=.1, levels=4, args={'varname': None}):
    '''Adaptive Function Approximation

    Progressivly refine a function if the average between two values is greater than err

    lower : lower bound of x
    upper : upper bound of x
    n : initial number of samples in the domain
    levels : number of refinements
    args = dict of arguments to pass to the function
        args['varname'] = name of the variable expected by the function
    Return
    x : domain variable
    y : f(x) range variable
    '''

    x = np.linspace(lower, upper, n)
    arguments = {}
    for key, arg in args.iteritems():
        if key != 'varname':
            arguments[key] = arg
        else:
            arguments[arg] = x
    y = f(**arguments)
    level = 0   # level of refinement
    y = np.reshape(y, x.shape)
    tot = np.sum(y[:-1] * (x[1:] - x[:-1]))

    while level < levels:
        #refinement loop
        rel_err = np.abs(((y[2:] + y[:-2]) / 2 - y[1:-1])) / tot
        idx = np.where(rel_err > err * np.max(tot))[0]

        new_x = (x[idx] + x[idx + 1]) / 2.
        arguments = {}
        for key, arg in args.iteritems():
            if key != 'varname':
                arguments[key] = arg
            else:
                arguments[arg] = new_x
        if new_x.shape[0] > 0:
            new_y = f(**arguments)
        else:
            new_y = np.array([])
            level = levels

        x = np.insert(x, idx + 1, new_x)
        y = np.insert(y, idx + 1, new_y)
        level += 1
    return x, y


def rolling_avg(x, window=2):
    '''Rolling Average'''
    weightings = np.repeat(1.0, window) / window
    return np.convolve(x, weightings)[window - 1:-(window - 1)]


class cpd(object):
    '''Conditional Probability Distribution'''
    cc = np.spacing(1)

    def __init__(self, data=None, *args, **kwargs):

        self.max_v = float('inf')
        self.min_v = -float('inf')
        self._data = data
        self._bounds = None
        self._ymean = None
        self._yscale = None
        pass

    @property
    def yscale(self):
        if self._yscale is None:
            if self._data is not None:
                self._yscale = np.std(self._data[0, :], 1)
            else:
                return 1.0
        return self._yscale

    @yscale.setter
    def yscale(self, value):
        self._yscale = value

    @property
    def ymean(self):
        if self._ymean is None:
            if self._data is not None:
                self._ymean = np.mean(self._data[0:1, :], 1)
            else:
                return 0.0
        return self._ymean

    @ymean.setter
    def ymean(self, value):
        self._ymean = value

    @property
    def stdbounds(self):
        '''Bounds of the scaled range'''
        bounds = self.bounds
        return (bounds - self.ymean) / self.yscale

    @stdbounds.setter
    def stdbounds(self, value):
        val = np.array(value)

        self.bounds = value * self.yscale + self.ymean

    @property
    def varnames(self):
        if not hasattr(self, '_varnames'):
            varnames = ['Y']
            try:
                xnames = ['X%s' % x for x in range(self._d_x)]
            except:
                xnames = []

            for name in xnames:
                varnames.append(name)
            return varnames
        else:
            return self._varnames

    @varnames.setter
    def varnames(self, value):
        self._varnames = value

    @property
    def bounds(self):
        '''Return the most restricting bounds in nonscaled range'''
        if self._bounds is None:
            self._bounds = np.array([self.min_v, self.max_v]) * self.yscale + self.ymean

        return self._bounds

    @bounds.setter
    def bounds(self, value):
        self._bounds = np.array(value, dtype='d')

    def __getstate__(self):
        '''Get State For Pickling'''
        return self.__dict__

    def __setstate__(self, _dict):
        '''Set State For Unpickling'''
        self.__dict__ = _dict

    def likelihood(self, Y, X=None):
        likelihood = self.evaluate(Y, X)
        return likelihood

    def sample(self, X=None, n=1, method=None):
        '''General Sampling'''
        bounds = self.bounds

        if self._d_x == 0:
            samples = self.kde.sample(X=X, n=n)
        else:
            scaleX2 = (X - self._xmean) / self._xscale

            mi = np.min(scaleX2, 1)
            ma = np.max(scaleX2, 1)

            samples = c_kcde.cdf_sample(scaleX2, self._yscale, self._ymean,
                                        self.s11, self.s12, self.s22i, self.U,
                                        self._w[0, :], self._u, bounds, self.alphah, n)
        return samples


    def loglikelihood(self):
        """Calculates the loglikelihood of the data.

        This method implements the log of the g function (equation 12) from:

        Cooper, Herskovitz. A Bayesian Method for the Induction of
        Probabilistic Networks from Data.
        """
        self.fit()
        return -self.score

    def fit(self, data=None, max_iter=None):
        '''Fit a kernel estimate to the data
        1D estimates are done with scipy gaussian kde
        Multidimensional estimates are done by LSCDE
        '''

        if data is not None:
            self.data = data
            data = self._data
        elif self._data is not None:
            data = self._data
        else:
            raise('data not set')
        if max_iter is None:
            max_iter = self.max_iter
        else:
            self.max_iter = max_iter

        if data.shape[0] == 1:
            self._d_y = d_y = 1
            self.n = n = data.shape[0]

        elif data.shape[0] > 1:
            self._d_y = d_y = 1
            self._d_x = d_x = data.shape[0] - d_y
            self.n = n = data.shape[1]

        if self._d_x == 0:
            self.max_v = max_v = np.max(self._scaled_data) + 3
            self.min_v = min_v = np.min(self._scaled_data) - 3
            self.kde = kde(self._data, self.bounds)
            ph_cv = self.evaluate(Y=self._data)
            #KL Loss
            self.score = -np.mean(np.log(ph_cv + self.cc))
        else:
            score = float('inf')
            old_score = float('inf')
            i = 1.
            while i <= max_iter:
                relax = (max_iter - i + 1) / (max_iter + 1)
                failed_counter = 0
                while score == float('inf') or score == old_score or (old_score ==
                                                                       float('inf')):
                    #FIXME Loop logic needs to be cleaned up. Awkward
                    if self.sigma is not None and self.lambd is not None:
                        sig = self.sigma
                        lam = self.lambd
                        old_score = score
                        self.sigma = ((1 + relax * np.linspace(-1, 1, 2)) *
                                      self.sigma)
                        self.lambd = ((1 + relax * np.linspace(-1, 1, 2)) *
                                      self.lambd)
                        alpha = np.sort(self.alphah, 0)
                        index = math.floor(self.number_basis * (.5 / (max_iter)
                                                * (i - max_iter) + 1))

                    score = self.LSCDE()
                    sigma_lambd_original = [self.sigma, self.lambd]

                    if score == float('inf'):
                        if old_score == float('inf'):
                            failed_counter += 1
                            self.lambd = self.lambd * 10
                        else:
                            self.sigma = sig
                            self.lambd = lam
                            self.score = score = old_score
                            break

                    if failed_counter > max_iter + 3:
                        self.plot()
                        raise Exception('Error in fitting node')
                if score <= old_score:
                    sigma = self.sigma
                    lambd = self.lambd
                    old_score = score
                else:
                    self.sigma = sig
                    self.lambd = lam
                    score = old_score
                i += 1
                self.alpha_h(self.sigma, self.lambd)

            if len(np.where(self.alphah>0)[0]) == 0:
                pass
            else:
                #Prune basis functions that are zero
                self.updateBasis(number_basis=np.count_nonzero(self.alphah),
                                 mask=np.where(self.alphah>0)[0])

    def quantile(self, quantiles=[.5], evidence=None, n=100,
                 levels=10, err=.01):
            '''Quantiles '''
            quantiles = np.atleast_1d(quantiles)
            ans = []
            for i in range(evidence.shape[0]):
                ccdf, y = self.cdf(evidence[[i]][:, np.newaxis], n, levels, err)

                quant = interp1d(ccdf, y)
                ans.append(quant(quantiles))

            ans = np.array(ans)
            return ans

    def cpdf(self, X=None, n=100, levels=5, err=.1):
        '''Conditional Probability Distribution Function evaluated at X'''
        bounds = self.bounds
        #1D
        if self._d_x == 0:
            Y, pdf = approx(self.evaluate, bounds[0], bounds[1], n, err=err,
                            levels=levels, args={'varname':'Y'})
        #Multidimensional
        else:
            #test to see if there is a sigma
            if self.alphah is None:
                raise Exception('LSCDE not run yet')
            bounds = self.bounds
            Y = np.linspace(bounds[0], bounds[1], n)
            pdf = self.evaluate(Y, X)[0]
            if any(np.isnan(pdf)):
                raise ValueError('NAN in pdf')
        binwidth = Y[1:] - Y[:-1]
        c = np.sum(binwidth * ((pdf[1:] + pdf[:-1]) / 2))
        pdf = pdf / c
        return pdf, Y  # likelihood, cumulative value, expected value

    def cdf(self, X=None, n=2000, levels=5, err=.1, Both=False, samp=10000):
        '''Cumulative Distribution Function'''
        t1 = time.time()
        pdf, Y = self.cpdf(X, n, 1, err)

        binwidth = Y[1:] - Y[:-1]
        cdf = np.cumsum(binwidth * ((pdf[1:] + pdf[:-1]) / 2))
        cdf = np.insert(cdf, 0, 0)
        cdf = cdf / cdf[-1]

        return cdf, Y  # likelihood, cumulative value, expected value

    def plot(self, evidence=None, fig=None, display=['data', 'hist', 'kde',
                             'basis', 'contours'], alphah=None, domain=None,
                             quantiles=None, quantileEvidence=None):
        '''Plot the data and condition density
        domain : [d_x,2] domain for plotting the contours
        '''
        num_vars = self._d_x + self._d_y
        plotshow = False
        if fig is None:
                fig = plt.figure(figsize=(3 * num_vars, 3 * num_vars))

##################################################################################
        #plot setup for no evidence variable
        if self._d_x == 0:
            ax = plt.gca()
            #plot base data
            if self._data is not None:
                if 'hist' in display:
                    ax.hist(self._data.T)
                #plot the probability distribution
                if 'kde' in display:
                    ph = self.cdf()
                    ax2 = ax.twinx()
                    bounds = self.bounds
                    ax2.plot(ph[1], ph[0])

################################################################################
        #plot setup for one dimension evidence variable
        if self._d_x == 1:
            ax = fig.add_subplot(111)
            bounds = self.bounds
            activeBasis = (self.alphah > 0)
            basisX = self._u[:, activeBasis] * self._xscale + self._xmean
            #plot data
            if 'data' in display:
                ax.plot(self._data[1, :], self._data[0, :], '.')
            #show basis functions
            if 'basis' in display:
                ax.plot(basisX, self._v[:, activeBasis] * self.yscale + self.ymean,
                        'o', color='0', markerfacecolor='None', markersize=15,
                        markeredgewidth=1)
                plt.ylim([bounds[0], bounds[1]])
            #plot contours
            if 'contours' in display:
                if domain is None:
                    domain = [basisX.min(), basisX.max()]
                n = 500
                x = np.linspace(domain[0], domain[1], n)
                y = np.linspace(self.bounds[0], self.bounds[1], n + 5)
                X, Y = np.meshgrid(x, y)
                Z = self.evaluate(y, x[np.newaxis, :])
                plt.contourf(x, y, Z, alpha=.3)
                plt.colorbar()

            #plot the conditional distribution
            if evidence is not None:
                evidence = np.atleast_1d(evidence)
                num_evidence = evidence.shape[0]
                for x in evidence:
                    ph, Y = self.cpdf(np.array([[x]]), n=500)

                    _scale = 1.0 / num_evidence
                    if self._u.shape[0] > 1:
                        cdf_scale = ((np.max(self._u) - np.min(self._u)) /
                                     np.max(ph) * self._xscale[0,:] * _scale)
                    elif domain is not None:
                        cdf_scale = (domain[1] - domain[0]) * _scale / np.max(ph)
                    else:
                        cdf_scale = ((bounds[1] - bounds[0]) * _scale /
                                      np.max(ph) / self._xscale[0, :])
                    ax.fill_betweenx(Y, x, x + ph * cdf_scale, alpha=.5, color='g')
            #Plot quantiles
            if quantiles is not None:
                if quantileEvidence is None:
                    quantileEvidence = np.linspace(bounds[0], bounds[1], 20)
                quantiles = np.atleast_1d(quantiles)
                eV = self.quantile(quantiles, quantileEvidence, n=1000)
                for q in range(quantiles.shape[0]):
                    d = 10 - 30 * (quantiles[q] - .5) ** 2
                    plt.plot(quantileEvidence, eV[:, [q]], 'k--',
                             label=str(quantiles[q] * 100) + '% Quantile',
                             linewidth=2, dashes=(d, 2 * d))

        #For higher dimensions
        if self._d_x > 1:
            datadict = OrderedDict(zip(self.varnames, self._data))
            scattermatrix(datadict, fig, display=display)
            if 'basis' in display:
                activeBasis = (self.alphah.T > 0)
                basisdict = OrderedDict(zip(self.varnames,
                                            self._data[:, activeBasis]))
                scattermatrix(basisdict, fig, display=display, marker='o')
        if plotshow == True:
            plt.show()

    def plotCDF(self, evidence=None):
        '''Plot Conditional Distribution
        '''
        fig = plt.figure()
        ax = fig.add_subplot(111)

        if evidence is not None:
            evidence = np.atleast_2d(evidence)
            for x in evidence:
                ph = self.cdf(x)
                bounds = self.bounds
                ax.plot(np.linspace(bounds[0], bounds[1], ph.shape[1]), ph[0, :])

        plt.show()

    def plotHist(self, evidence=None, fig=None):
        '''Plot a histogram and pdf'''
        if fig is None:
            plt.figure()
#         testX = np.array([[0.25]])
        bounds = self.bounds
        testX = (evidence - self._xmean) / self._xscale
        samples = c_kcde.cdf_sample(testX, self._yscale, self._ymean, self.s11,
                                    self.s12, self.s22i, self.U, self._w[0, :],
                                    self._u, bounds, self.alphah, 100)

        ax = plt.gca()
        pdf, Y = self.cpdf(evidence, n=1000)
        plt.plot(Y, pdf)
        ax2 = ax.twinx()
        plt.hist(samples, normed=True, alpha=.5)
        ax.set_xlabel(evidence)


class anisokcde(cpd):
    def __init__(self, data=None, config={}, *args, **kwargs):
        """Conditional Density Estimate Based on Anisotropic Gaussian Kernel
            * Example Formatting
                - 2nd level
        """
        super(anisokcde, self).__init__(*args, **kwargs)
        #precision
        self.dtype = np.float96
        #validation setup

        self.validation = 'kfold'  # kfold or test
        self.folds = 5
        self._number_basis = 300
        self.loss_type = "KL"  # Either 'KL' or 'SQ'

        self.logging = 2  # c ontrol the logging

        self.sigma = None
        self.lambd = None
        self.score = None

        self._u = None
        self._v = None
        self.max_v = float('inf')
        self.min_v = -float('inf')

        self.alphah = None
        self.basis_index = None
        self._neighborhood = None
        self._epsilon = .1  # size of the neighborhood
        self._cov = None
        self.M = None
        self.U = None
        self.detW0 = None

        self._d_y = 1  # a data setter would update this
        self._d_x = 0  # a data setter would update this

        self.max_iter = 10
#         self.data = data #set data

        for key, value in config.iteritems():  # take input config dictionary
            try:  # if attribute already exists. Protects setters and getters
                self.__setattr__(key, value)
            except AttributeError:
                self.__dict__[key] = value  # new key
        self.data = data  # set data

    @property
    def epsilon(self):
        return self._epsilon

    @epsilon.setter
    def epsilon(self, value):
        self._epsilon = value
        self._cov = None

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, value):
        '''Set data and Standardize'''
        self._d_y = d_y = 1
        self._d_x = d_x = value.shape[0] - d_y

        self.n = value.shape[1]  # number of data points

        ytrain = value[0:d_y, :]
        xtrain = value[d_y:d_y + d_x, :]

        #standardization
        self._xscale = xscale = np.atleast_2d(np.std(xtrain, 1)).T
        self._xmean = xmean = np.atleast_2d(np.mean(xtrain, 1)).T
        if xscale.any() == 0:
            self._xscale = xscale = np.ones([d_x, 1])

        self.yscale = yscale = np.std(ytrain)  # *0+1
        self.ymean = ymean = np.mean(ytrain)  # *0
        if yscale == 0:
            self._yscale = yscale = np.ones([d_y, 1])
        x_train = (xtrain - xmean) / xscale
        y_train = (ytrain - ymean) / yscale

        self._data = value
        self._scaled_data = np.vstack([y_train, x_train])

        self.neighborhood = mypebl.distributions.c_kcde.neighborhood()
        self.neighborhood.data = self._scaled_data.T
        self.updateBasis(number_basis=self.number_basis)

    @property
    def number_basis(self):
        return self._number_basis

    @number_basis.setter
    def number_basis(self, value):
        self._number_basis = value
        self.updateBasis(number_basis=value)

    def updateBasis(self, number_basis=None, mask=None):
        ## Select Basis
        if hasattr(self, 'data') and hasattr(self, 'n'):
            if mask is None:
                # Number of kernel bases
                self._number_basis = b = min(number_basis, self.number_basis, self.n)
                ####### Choose kernel centers w=(u,v) for z=(x,y)
                #kmeans to selected basis locations
                cluster = KMeans(n_clusters=self._number_basis,  n_jobs=1)
                cluster.fit(self._scaled_data.T)
                self.basis_index = np.ravel(self.neighborhood.tree.query(
                                         cluster.cluster_centers_, return_distance=False))
                self.alphah = 1. / b * np.ones([b, 1])
            else:
                if number_basis is not None:
                    self._number_basis = number_basis
                b = max(number_basis, mask.shape[0])
                newBasis = abs(b - self.number_basis)
                if newBasis > 0:
                    if self.n > self.number_basis:
                        #add new basis
                        oldBasis = set(self.basis_index)
                        totalN = set(range(self.n))
                        possibleBasis = totalN.difference(oldBasis)
                        addBasis = np.random.permutation(list(possibleBasis))[0:newBasis]

                        mask = np.hstack([self.basis_index[mask], addBasis])
                        self.basis_index = mask

                else:
                    #No new basis function needed only prune
                    self.alphah = self.alphah[mask]
                    self._cov = self._cov[mask]
                    self.U0 = self.U0[mask]
                    self.Wi = self.Wi[mask]
                    self.U = self.U[mask]
                    self.M = self.M[mask]
                    self.V = self.V[mask]
                    self.W0 = self.W0[mask]
                    self.detW0 = self.detW0[mask]
                    self.sig = self.sig[mask]
                    self.s11 = self.s11[mask]
                    self.s12 = self.s12[mask]
                    self.s22i = self.s22i[mask]
                    self.basis_index = self.basis_index[mask]

            y_train = self._scaled_data[0:self._d_y, :]
            x_train = self._scaled_data[self._d_y:self._d_y + self._d_x, :]
            self._u = u = x_train[0:, self.basis_index]
            self._v = v = y_train[0:, self.basis_index]
            self._w = np.vstack([v, u])
            self._z = np.vstack([y_train, x_train])

            self.min_v = np.min(self._v) - 3 * self._yscale
            self.max_v = np.max(self._v) + 3 * self._yscale
            self._number_basis = self.basis_index.shape[0]
            self.neighborhood.basis_index = self.basis_index

    @property
    def distances(self):
        data_frac = min(100, np.ceil(.6 * self._z.shape[1]))
        XA = self._z[:, 0:data_frac].T

        dists = cdist(XA, XA)  # euclidean distances
        return dists[np.triu_indices(dists.shape[0], 1)]

    @property
    def cov(self):
        '''Local Covariance for the basis functions using their nearest neighbors'''
        if self._cov is None:
            (self.U0, self.Wi, self.U, self.M, self.V, self.W0, self.detW0,
             self._cov, self.s11, self.s12, self.s22i) = mypebl.distributions.c_kcde.cov(
                                                    self._scaled_data, self.basis_index,
                                                    self.neighborhood, self.epsilon)
            self.sig = self.W0
        return self._cov

    def LSCDE(self, data=None):
        if self.sigma is None:
            dists = self.distances
            min_d = np.min(dists)
            max_d = np.max(dists)
            drange = max_d - min_d
            lower = min_d or .01 * drange
            upper = max_d - 0.5 * drange

            sigma_list = np.logspace(np.log10(lower), np.log10(upper), 7)
            sigma_list = np.logspace(np.log10(lower), np.log10(upper), 7)
        else:
            sigma_list = self.sigma
        if self.lambd is None:
            # Candidates of regularization parameter
            lambda_list = np.logspace(-1, 1.25, 5)
        else:
            lambda_list = self.lambd
        num_sigma = sigma_list.size
        num_lambda = lambda_list.size

        #Standardize data
        if data is not None:
            self.data = data
        elif self.data is None:
            raise Exception('Data Not Set')

        n = self.n
        d_y = self._d_y
        d_x = self._d_x

        y_train = self._scaled_data[0:d_y, :]
        x_train = self._scaled_data[d_y:d_y + d_x, :]

        ################################

        fold = self.folds  # Number of folds of cross-validation
        cv_fold = np.r_[0: fold]
        cv_index = np.random.permutation(n)
        cv_split = np.floor(np.r_[0:n] * fold / n)

        if self.validation == 'kfold':
            fold = self.folds

        elif self.validation == 'test':
            fold = 1  # only use  one fold as validation

        cc = np.spacing(1)
        w = self._w
        z = self._z
        b = self.number_basis
        score_cv = np.zeros([num_sigma, num_lambda])
        Phibar_cv = np.empty([b, b, self.folds])
        dt = time.time()
        ################################START HyperParm Sweep#################
        for sigma_index in range(sigma_list.size):
            self.sigma = sigma = sigma_list[sigma_index]
            self.epsilon = sigma
            self.cov

            qx_dist = dist.mahalanobis2_blas(x_train.T, self._u.T, self.U).T
            WW = 1.0 / (self.Wi[:, np.newaxis] + self.Wi[:, np.newaxis].T)
            W0 = np.tile(self.W0[:, np.newaxis], [1, self.number_basis])

            tmp2 = np.sum((self.V[:, 0] * self._u.T), 1)
            vv = -1 / self.W0[:, np.newaxis] * tmp2[:, np.newaxis] - self._v.T
            q = (vv - vv.T)
            v_dist = q * q * WW

            Wl = self.W0[:, np.newaxis] + self.W0[:, np.newaxis].T
            sqrtWl = np.sqrt(Wl)
            c = math.sqrt(math.pi / 2.) / sqrtWl * 2.

            w_bar = self.W0[:, np.newaxis] * vv
            mu = (w_bar + w_bar.T) / Wl
            lower, upper = self.stdbounds

            c = math.sqrt(math.pi / 2.) / sqrtWl * (erf((upper - mu) * sqrtWl /
                    math.sqrt(2.)) - erf((lower - mu) * sqrtWl / math.sqrt(2.)))
            p = (erf((upper - mu) * sqrtWl / math.sqrt(2.)) -
                 erf((lower - mu) * sqrtWl / math.sqrt(2.)))

            phi_qx = np.exp(-0.5 * qx_dist)
            phi_vv = np.exp(-0.5 * v_dist)

            phi_zw = np.exp(-0.5 * dist.mahalanobis2_blas(z.T, w.T, Vi=self.M)).T

            #K-fold Precalc Phibar
            for k in range(self.folds):
                tmp = phi_qx[:, cv_index[cv_split == k]]
                Phibar_cv[:, :, k] = c * phi_vv * np.dot(tmp, tmp.T)

            #Lambda Loop
            for lambda_index in range(lambda_list.size):
                lambd = lambda_list[lambda_index]
                score_tmp = np.zeros([1, fold])
                for k in range(fold):
                    H_hat = (np.sum(Phibar_cv[:, :, cv_fold != k], 2) /
                             sum(cv_split != k) + lambd * np.eye(b))
                    h_hat = np.mean(phi_zw[:, cv_index[cv_split != k]], 1)
                    try:
                        alphat_cv = np.linalg.solve(H_hat, h_hat)

                        alphah_cv = np.maximum(0, alphat_cv)
                        normalization_cv = self.normalization(alphah_cv,
                                    phi_qx[:, cv_index[cv_split == k]],
                                    x_train[:, cv_split == k])
                        ph_cv = np.dot(alphah_cv, phi_zw[:,
                                    cv_index[cv_split == k]]) / normalization_cv
                        score_tmp[0, k] = -np.mean(np.log(ph_cv + cc))
                    except Exception as E:
                        print E
                        score_cv[sigma_index, lambda_index] = float('nan')
                        break

                cond_num = np.linalg.det(H_hat)
                mean_score = np.mean(score_tmp)
                # penalize anwers with poor condition numbers
                if abs(cond_num) < 1e-20 or np.isnan(cond_num):
                    mean_score = float('inf')
                score_cv[sigma_index, lambda_index] = mean_score

        dt2 = time.time() - dt

        mylogger.info('Score %s' % score_cv)

        a = np.nanargmin(score_cv)
        lambda_index = a % lambda_list.size
        sigma_index = math.floor(a / lambda_list.size)

        if not math.isnan(sigma_index) and not math.isnan(lambda_index):
            self.lambd = lambd = lambda_list[lambda_index]
            self.sigma = sigma = sigma_list[sigma_index]
            self.score = score_cv[sigma_index, lambda_index]
        else:
            self.score = float('inf')

        if self.logging > 0:
            mylogger.debug('sig list %s' % sigma_list)
            mylogger.debug('lam list %s' % lambda_list)
            mylogger.debug('%s, %s, score: %s lambda %s sigma %s number basis %s '%(
                                 sigma_index, lambda_index,
                                 score_cv[sigma_index, lambda_index], lambd,
                                 sigma, np.count_nonzero(self.alphah)))
        return score_cv[sigma_index, lambda_index]

    def alpha_h(self, sigma, lambd):
        '''Solve Basis function weights
        sigma - standard deviation of the kernel
        lambd - stability factor
        '''
        self.epsilon = sigma
        self.cov
        ##Solve for final alpha
        d_y = self._d_y
        d_x = self._d_x
        y_train = self._scaled_data[0:d_y, :]
        x_train = self._scaled_data[d_y:d_y + d_x, :]

        qx_dist = dist.mahalanobis2_blas(x_train.T, self._u.T, Vi=self.U).T
        WW = 1.0 / (self.Wi[:, np.newaxis] + self.Wi[:, np.newaxis].T)
        W0 = np.tile(self.W0[:, np.newaxis], [1, self.number_basis])

        tmp2 = np.sum((self.V[:, 0] * self._u.T), 1)
        vv = -1 / self.W0[:, np.newaxis] * tmp2[:, np.newaxis] - self._v.T
        q = (vv - vv.T)
        v_dist = q * q * WW

        W1 = self.W0[:, np.newaxis] + self.W0[:, np.newaxis].T
        sqrtW1 = np.sqrt(W1)
        c = math.sqrt(math.pi / 2.) / sqrtW1 * 2
        w_bar = self.W0[:, np.newaxis] * vv
        mu = (w_bar + w_bar.T) / W1
        lower, upper = self.stdbounds
        c = math.sqrt(math.pi / 2.) / sqrtW1 * (erf((upper - mu) * sqrtW1 /
                math.sqrt(2.)) - erf((lower - mu) * sqrtW1 / math.sqrt(2.)))

        self.c = c
        self.v_dist = v_dist
        #FIXME: Exponent correct
        phi_qx = np.exp(-0.5 * qx_dist)
        self.phi_vv = phi_vv = np.exp(-0.5 * v_dist)

        phi_zw = np.exp(-0.5 * mahalanobis2(self._z, self._w, Vi=self.M))
        Phibar = self.c * phi_vv * np.dot(phi_qx, phi_qx.T)
        H_hat = Phibar / self.n + lambd * np.eye(self.number_basis)  # h_hat
        h_hat = np.mean(phi_zw, 1)  # H_hat

        alphat = np.linalg.solve(H_hat, h_hat)
        self.alphah = np.maximum(0, alphat)

    def normalization(self, alpha, phi_qx, scaleX):
        Const2 = c_kcde.normalization(alpha, phi_qx, scaleX, self.Wi,
                                      self.V[:, 0, :], self._u, self._v[0, :],
                                      self.number_basis, self.stdbounds,
                                      self.sig)

        return Const2

    def evalPhi(self, scaleX, scaleY):
        scaleX = np.atleast_2d(scaleX)
        scaleY = np.atleast_2d(scaleY)
        if scaleX.shape[1] == scaleY.shape[1]:
            if self._d_x > 1:
                pass
            phi = kern(scaleY, scaleX, self._w, Vi=self.M)
        else:
            raise Exception('Dim Mismatch  (X:%s, Y:%s)' % (scaleX.shape, scaleY.shape))
        Phi = np.dot(self.alphah.T, phi)[0]
        return Phi

    def evaluate(self, Y=None, X=None):
        '''Evaluate The Density
        Y = [number of Y] vector of Y values
        X = [X dimensions, number of X]
        '''

        if X is None and self._d_x == 0:
            s = self.kde.evaluate(Y)
            return s

        X = np.asanyarray(X, np.float64)
        Y = np.asanyarray(Y, np.float64)
        if not X.flags['C_CONTIGUOUS']:
            X = np.ascontiguousarray(X, dtype=np.float64)
        if not Y.flags['C_CONTIGUOUS']:
            Y = np.ascontiguousarray(Y, dtype=np.float64)
        a = np.array(c_kcde.evaluate(Y, X, self.alphah,  self.Wi,
                                     self.V[:, 0, :], self.U, self.M, self.sig,
                                     self._w, self._v[0, :], self._u,
                                     self._ymean, self._yscale,
                                     self._xmean[:, 0], self._xscale[:, 0],
                                     self.stdbounds))
        return np.nan_to_num(a)

    def invserse(self, y):
        if self._d_x == 1:
            if not hasattr(self, '_evalInverse'):
#                basisSort = np.sort([self._v[0], self._u[0]])
                indexSort = np.argsort(self._v[0])

                yBasis = self._v[0, indexSort]
                xBasis = self._u[0, indexSort]
                self._evalInverse = interp1d(yBasis, xBasis,
                                             bounds_error=False, fill_value=0)
        else:
            raise ValueError
        stdy = (y - self.ymean) / self.yscale
        return self._xscale[0] * self._evalInverse(stdy) + self._xmean[0]


def weight(self, X):
    '''Basis weights at scaled location X'''
    phi_qx = np.exp(-0.5 * dist.mahalanobis2_blas(X, self._u.T, self.U))
    pval = self.alphah * phi_qx[0]
    tot = np.sum(pval)
    kval = pval / tot
    return kval
```

```


def cdf_sample(self, X, n=50000):
    yscale = self._yscale
    ymean = self._ymean
    kval = weight(self, X)
    k = np.random.multinomial(n, kval)

    samples = np.array([])
    for i in range(self.number_basis):
        n = k[i]
        if n > 0:
            s11 = self.cov[i, 0, 0]
            s12 = self.cov[i, [0], self._d_y:]
            s22 = self.cov[i, self._d_y:, self._d_y:]
            s22i = np.linalg.inv(s22)
            mu = self._w[0, i] + np.dot(np.dot(s12, s22i),
                                        (X - self._u[:, i]))[0, 0]
            s = s11 - np.dot(s12.T, np.dot(s22i, s12))[0, 0]
            mu = mu * yscale + ymean
            s = s ** .5 * yscale
            while n > 0:
                sample = np.random.normal(mu, s, n)
                ind = np.where(sample < self.bounds[1])[0]
                ind = np.where(sample[ind] > self.bounds[0])[0]
                n -= len(ind)
                samples = np.hstack([samples, sample[ind]])
    return samples
```

A.3 Cython Helper Functions

```
#cython: embedsignature=True
#cython: boundscheck=False
#cython: wraparound=False
#cython: cdivision=True
#cython: profile=True

import numpy as np
import math
import time
from scipy.spatial.distance import cdist
from sklearn.neighbors import BallTree, radius_neighbors_graph, kneighbors_graph
from myMDAO.lib.plot.api import scattermatrix
import matplotlib.pyplot as plt
import time
from scipy.integrate import ode
import warnings
from itertools import izip

cimport cython
cimport numpy as np
from cython.parallel import prange, parallel, threadid
from libc.math cimport erf, sqrt
from ceygen.lu cimport det, inv, iinv
from ceygen.core cimport dot_vv, dot_mv, dot_vm, dot_mm
from ceygen.elemwise cimport add_vs, multiply_ms, multiply_vs

from libc.stdlib cimport malloc, free
from libc.math cimport exp, sqrt

from cython.view cimport array
cimport random
cimport cblas
cimport dist

np.import_array()

```

```
DTYPE = np.float64
ctypedef np.float64_t DTYPE_t

ITYPE = np.int32
ctypedef np.int32_t ITYPE_t

cpdef np.ndarray[DTYPE_t, ndim=2] evalPhi(double [:] Y, double [:,:] X,
                                          double [:] alphah, double [:,::1] W,
                                          double [:,:,::1] Vi):

    cdef:
        int d_x, nX, n, k
        int d_y = 1
        int nY = Y.shape[0]

        int i, j, c, l
        int d = W.shape[0]
        int b = W.shape[1]
        np.ndarray[DTYPE_t, ndim = 2] z
        np.ndarray[DTYPE_t, ndim = 2] phi
        double [:, ::1] Wt
        double [:, :] Phi2

    if X is not None:
        d_x = X.shape[0]
        nX = X.shape[1]
        if nX == nY:
            n = 1
        else:
            n = nX
    else:
        d_x = 0
        nX = 1
        n = 1
    assert d == d_x + 1
    assert d == Vi.shape[1]
    assert d == Vi.shape[2]
    assert b == Vi.shape[0]
    Phi2 = np.empty([n, nY], dtype=DTYPE)

    z = np.empty([nY, d_x + 1], dtype=DTYPE, order='c')
    Wt = np.asarray(W.T, dtype=DTYPE, order='c')
    phi = np.zeros([nY, b], dtype=DTYPE, order='c')

    for i in range(nY):
        z[i, 0] = Y[i]
    for j in range(n):
        if nX == nY:
            for i in range(nX):
                for k in range(d_x):
                    z[i, k + 1] = X[k, i]
        else:
            for k in range(d_x):
                z[:, k + 1] = X[k, j]
        dist.mahalanobis_cdist(z, Wt, Vi, phi)
        for k in range(nY):
            for l in range(b):
                phi[k, l] = exp(-.5 * phi[k, l])
        dot_vm(alphah, phi.T, Phi2[j])
    return np.asarray(Phi2.T, order='c')


cpdef double[:,::1] phi_qx(double [:,:] X, double [:,:] ux, double [:,:,::1] U):

    cdef:
        double [:, ::1] phi
        double [:, ::1] retPhi
        int b = U.shape[0]
        int num_d = U.shape[1]

        double [:,::1] uxt = np.asarray(ux.T, dtype=DTYPE, order='c')
        double [:,::1] Xt = np.asarray(X.T, dtype=DTYPE, order='c')
        int num_X

    assert num_d == U.shape[2]
    assert num_d == Xt.shape[1]
    assert num_d == uxt.shape[1]
    assert b == uxt.shape[0]

    num_X = Xt.shape[0]

    phi = np.zeros([num_X, b], dtype=DTYPE, order='c')
    retPhi = np.zeros([b, num_X], dtype=DTYPE, order='c')
    dist.mahalanobis_cdist(Xt, uxt, U, phi)

    for j in range(num_X):
        for i in range(b):
            retPhi[i, j] = exp(-.5 * phi[j, i])
    return retPhi


def cdf_sample(double[:,:] x, double yscale, double ymean, double [:] s11,
               double [:,:,:] s12, double [:,:,:] s22i, double [:,:,::1] U,
               double [::1] _w, double[:,:] _u, double [:] bounds, double [::1] alphah,
               int num_samples=1):
    cdef:
        int num_basis = _w.shape[0]
        unsigned int *k = <unsigned int *>malloc(num_basis * sizeof(unsigned int))
        int n, i, j, c
        int d = x.shape[0]
        int num_X = x.shape[1]
        double [:] q = np.empty(d)
        np.ndarray[DTYPE_t, ndim=2] samples
        double sigma, mu, s, tot = 0, tmp, lower, upper

        random.gsl_rng *r = random.gsl_rng_alloc(random.gsl_rng_mt19937)

        double [:, ::1] qx
        double [:] kval = np.empty(num_basis)
        double [:] tmp2 = np.empty(d)
        double [:] tmp3 = np.empty(d)
        double [:, :] si

    assert num_basis == alphah.shape[0]
    assert num_basis == s11.shape[0]
    assert d == _u.shape[0]
    assert num_basis == _u.shape[1]
    assert d == x.shape[0]
    assert num_basis == U.shape[0]
    assert d == U.shape[1]
    assert d == U.shape[2]
    assert d == _u.shape[0]
    assert num_basis == _u.shape[1]

    if num_samples == num_X:
        num_samples = 1
    samples = np.empty([num_samples, num_X])
    qx = phi_qx(x, _u,  U)

    for c in range(num_X):
        tot = 0
        for l in range(num_basis):
            tmp = alphah[l] * qx[l, c]
            kval[l] = tmp
            tot += tmp

        for l in range(num_basis):
            kval[l] = kval[l] / tot

        random.multinomial(r, kval.shape[0], num_samples, &kval[0], &k[0] )
        j = 0

        for l in range(num_basis):
            n = k[l]
            if n > 0:
                for i in range(d):
                    q[i] = x[i, c] - _u[i, l]
                    tmp3[i] = s12[l, 0, i]
                si = s22i[l]
                dot_vm(tmp3, si, tmp2)
                mu = _w[l] + dot_vv(tmp2, q)
                dot_mv(si, tmp3, tmp2)
                s = s11[l] - dot_vv(tmp3, tmp2)

                mu = mu * yscale + ymean
                sigma = sqrt(s) * yscale

                while n > 0:
                    sample = random.gaussian(r, sigma) + mu

                    if (sample < bounds[1]) and (sample > bounds[0]):
                        samples[j, c] = sample  #save the points between the bounds
                        n -= 1
                        j += 1
                    else:
                        #Missed
                        if (mu < bounds[0]) or (mu > bounds[1]):
                            lower = (bounds[0] - mu) / sigma
                            upper = (bounds[1] - mu) / sigma
                            while n > 0:
                                sample = truncatedStdNorm(r, lower, upper)
                                samples[j, c] = (sample * sigma + mu)
                                n -= 1
                                j += 1
    random.gsl_rng_free(r)
    free(k)
    return samples


cdef double truncatedStdNorm(random.gsl_rng *r, double lower, double upper) nogil:
    'Generate n samples from Truncated Standard Normal Distribtion between lower and upper'
#    assert lower<upper
    cdef:
        double sample
        double u, p, z
        int n = 0

    while n < 1:
        z = random.uniform(r, lower, upper)
        if upper < 0:
            p = exp((upper ** 2 - z ** 2) / 2.)
        elif lower > 0:
            p = exp((lower ** 2 - z ** 2) / 2.)
        else:
            p = exp(-.5 * z ** 2)
        u = random.uniform(r, 0, 1)

        if u <= p:  # accept Rate
            return z


def evaluate(double [::1] Y, double [:, ::1] X,
             double [:] alphah,  double [:] Wi, double [:, :] V,
             double [:, :, ::1] U, double [:, :, ::1] M, DTYPE_t [:] sig,
             double [:, ::1] _w, double [:] _v, double [:, :] _u,
             double ymean, double yscale, double [:] xmean, double[:] xscale,
             DTYPE_t [:] bounds):
    cdef:
        int dx = X.shape[0]
        int num_x = X.shape[1]
        int num_y = Y.shape[0]  # number of y locations
        int num_basis = alphah.shape[0]
        int i, j, c, b
        np.ndarray phi
        double [:, ::1] tmp
        double [:, ::1] qx
        double [:, ::1] Phi
        double [::1] scaleY = np.zeros(num_y)
        double [:, ::1] scaleX = np.zeros([dx, num_x])
        double [:, :] scaleX2 = np.zeros([dx, 1])
        double [:, :] qx2 = np.zeros([num_basis, 1])
        double [:] const = np.zeros(num_x)

    assert dx == xscale.shape[0]
    assert dx == xmean.shape[0]

    for i in range(num_y):
        scaleY[i] = (Y[i] - ymean) / yscale  # goes from 1d to 2d here. Needed?

    for i in range(dx):
        for j in range(num_x):
            scaleX[i, j] = (X[i, j] - xmean[i]) / xscale[i]
    qx = phi_qx(scaleX, _u, U)

    Phi = evalPhi(scaleY, scaleX, alphah, _w, M)

    const = normalization(alphah, qx, scaleX, Wi, V, _u, _v, num_basis, bounds, sig)

    for i in range(Phi.shape[0]):
        for j in range(Phi.shape[1]):
            Phi[i, j] = Phi[i, j] / const[j]

    if num_y == Phi.shape[1] or num_x == 1:
        return Phi.T
    elif num_y == Phi.shape[0]:
        return Phi


cdef void covariance_matrix(double [:,::1] X_buf, double[:,::1] out):
    """Algorithms for computing the sample variance: Analysis and recommendations"""
    cdef:
        unsigned int i, j, d = X_buf.shape[0], n = X_buf.shape[1]
        double mean, s
        unsigned int LDA, LDB, LDC, M, N, K
    out[:, :] = 0

    M = d  # Number of rows in matrices A and C.
    N = d  # Number of columns in matrices B and C.
    K = n  # Number of columns in matrix A; number of rows in matrix B.
    LDA = n
    LDB = n
    LDC = d

    for i in range(d):
        mean = 0
        for j in range(n):
            mean += X_buf[i, j]

        s = -mean / n
        add_vs(X_buf[i], s, X_buf[i])
        mean = 0

    cblas.dgemm(cblas.CblasRowMajor, cblas.CblasNoTrans, cblas.CblasTrans, M,
                N, K, 1.0, &X_buf[0,0], LDA, &X_buf[0,0], LDB, 0.0, &out[0,0], LDC)
    if n > 2:
        s = 1.0 / (n - 1.0)
        for i in range(d):
            for j in range(d):
                out[i, j] = s * out[i, j]


def cov(np.ndarray[double, ndim=2] data, np.ndarray[int, ndim=1] basis_index,
        neighborhood, epsilon, weighted=True):
        '''Local Covariance for the basis functions using their nearest neighbors'''
        cdef:
            unsigned int number_basis = basis_index.shape[0]  # input
            unsigned int l, i, b, j, d_y, d_x, d, u, done, num_indices
            unsigned int num_data, neighbors
            double s = .05  # used to streach a singular covariance matrix
            double cov_det
        d_y = 1  # self._d_y number y dimensions
        d = data.shape[0]  # number x dimensions
        num_data = data.shape[1]  # number of data points
        d_x = d - d_y  # number problem dimensions

        #print 'basis #',number_basis

        #numpy version
        cdef:
            np.ndarray[double, ndim=3] np_M = np.empty(dtype='d', shape=(number_basis, d, d))
            np.ndarray[double, ndim=1] np_W0 = np.empty(dtype='d', shape=(number_basis))
            np.ndarray[double, ndim=3] np_U0 = np.empty(dtype='d', shape=(number_basis, d_x, d_x))
            np.ndarray[double, ndim=3] np_V = np.empty(dtype='d', shape=(number_basis,d_y,d_x))
            np.ndarray[double, ndim=3] np_Vt = np.empty(dtype='d', shape=(number_basis, d_x, d_y))
            np.ndarray[double, ndim=3] np_U = np.empty(dtype='d', shape=(number_basis, d_x, d_x))
            np.ndarray[double, ndim=1] np_Wi = np.empty(dtype='d', shape=(number_basis))
            np.ndarray[double, ndim=3] np_cov = np.empty(dtype='d', shape=(number_basis, d, d))
            np.ndarray[double, ndim=2] np_covariance = np.empty(dtype='d', shape=(d, d))
            np.ndarray[double, ndim=2] np_tmp = np.empty(dtype='d', shape=(d, d))
            np.ndarray[double, ndim=2] np_tmp2 = np.empty(dtype='d', shape=(d_x, d_x))
            np.ndarray[double, ndim=1] np_detW0 = np.empty(dtype='d', shape=(number_basis))

            np.ndarray[double, ndim=1] np_s11 = np.empty(dtype='d', shape=(number_basis))
            np.ndarray[double, ndim=3] np_s12 = np.empty(dtype='d', shape=(number_basis, d_y, d_x))
            np.ndarray[double, ndim=2] np_s22 = np.empty(dtype='d', shape=(d_x, d_x))
            np.ndarray[double, ndim=3] np_s22i = np.empty(dtype='d', shape=(number_basis, d_x, d_x))

        cdef:
            double [:, :, :] M = np_M
            double [:] W0 = np_W0
            double [:, :, :] U0 = np_U0
            double [:, :, :] V = np_V
            double [:, :, :] Vt = np_Vt
            double [:, :, :] U = np_U
            double [:] Wi = np_Wi
            double [:, :, :] _cov = np_cov
            double [:, ::1] covariance = np_covariance
            double [:, :] tmp = np_tmp
            double [:, :] tmp2 = np_tmp2
            double [:] detW0 = np_detW0

            double [:, ::1] X_buf = np.empty(dtype='d', shape=(data.shape[0], data.shape[1]), order='C')
            double [:] s11 = np_s11
            double [:, :, :] s12 = np_s12
            double [:, :] s22 = np_s22
            double [:, :, :] s22i = np_s22i

        if weighted:
            neighborhood_indices = neighborhood.query(100)
        else:
                neighborhood_indices = neighborhood.query(epsilon)
            for l in xrange(number_basis):
                indices = neighborhood_indices[l]

                if weighted:
                    b = basis_index[l]
                    indices = [i for i in indices if i != b]
                    X = data[:, indices]
                    X -= data[:, [b]]
                    distances = np.sum(X ** 2, 0) ** .5
                    weights = np.exp(-.5 * distances / epsilon)
                    X = weights * X
                    covariance = np.dot(X, X.T) / np.sum(weights)
                else:
                    num_indices = len(indices)
                    X_buf = data[:, indices].copy()
                    covariance_matrix(X_buf[:, :num_indices], covariance)
                    cov_det = det(covariance)
                    done = 0
                    neighbors = 1 + num_indices
                    i = 0
                    while i <= d and neighbors <= num_data:
                        #Verify diagonal does not have nonzero entries
                        if covariance[i, i] == 0:
                            #add more nearest neighbors
                            indices = neighborhood.query(epsilon, basis=l, k=neighbors)[0]
                            num_indices = len(indices)
                            X_buf = data[:, indices].copy()
                            covariance_matrix(X_buf[:, :num_indices], covariance)
                            neighbors += 1
                            i = 0
                        else:
                            i += 1
                while det(covariance) < 1e-10 and done < 10:  # det close to zero, singular covariance

                    for i in range(0,d):
                        covariance[i,i] += s * covariance[i,i]
                    done = done + 1

                s11[l] = covariance[0, 0]
                s12[l] = covariance[0, d_y:]
                s22 = covariance[d_y:, d_y:]
                s22i[l] = inv(s22, tmp2)

                _cov[l] = covariance

                inv(_cov[l], tmp)
                M[l, :, :] = tmp[:, :]
                W0[l] = M[l, 0, 0]
                U0[l] = M[l, d_y:, d_y:]
                V[l] = M[l, :d_y, d_y:]
                Vt[l] = M[l, d_y:, :d_y]
                Wi[l] = 1 / W0[l]

                tmp[:, :] = 0
                cblas.dger(cblas.CblasColMajor, d, d, 1, &Vt[l, 0, 0], 1,
                           &V[l, 0, 0], 1, &tmp[0, 0], d)
                for i in range(d_x):
                    for j in range(d_x):
                        U[l, i, j] = U0[l, i, j] - Wi[l] * tmp[i, j]
                detW0[l] = sqrt(W0[l])
            return np_U0, np_Wi, np_U, np_M, np_V, np_W0, np_detW0, np_cov, np_s11, np_s12, np_s22i


@cython.initializedcheck(False)
def normalization(double [:] alpha, double [:,:] phi_qx, double [:,:] scaleX,
                  double [:] Wi, double [:,:] V, double [:,:] _u, double [:] _v,
                  ITYPE_t number_basis, DTYPE_t [:] bounds, DTYPE_t [:] sig):
```

```
    cdef:
        double pi = 3.14159265359
        unsigned int n, d, l = 0, i = 0, j = 0
        double red, s, mu, lower, upper, c
        double [:,:] C
        double cc = 2.22044604925e-16
        np.ndarray[DTYPE_t, ndim=1] Const2
    lower = bounds[0]
    upper = bounds[1]
    n = scaleX.shape[1]
    d = scaleX.shape[0]
    C = np.empty([number_basis, n])
    Const2 = np.empty(n)
    assert d == _u.shape[0]
    assert d == V.shape[1]
    assert n == phi_qx.shape[1]

    for l in prange(number_basis, nogil=True):
        for i in xrange(n):
            red = 0
            for j in xrange(d):
                red = red + V[l, j] * (scaleX[j, i] - _u[j, l])

            mu = _v[l] - Wi[l] * red

            s = sqrt(sig[l])
            c = (sqrt(pi / 2.) / s * (erf(s * (mu - lower) / sqrt(2.)) -
                                      erf(s * (mu - upper) / sqrt(2.))))
            C[l, i] = c * phi_qx[l, i]

    #finally matrix multiply
    dot_vm(alpha, C, Const2)

    for i in range(n):
        if Const2[i] < cc:
            Const2[i] = cc
    return Const2


class neighborhood(object):
    def __init__(self, data=np.empty(0), b_index=None, leaf_size=500):
        self.leaf_size = leaf_size
        self.data = data.T
        self.basis_index = b_index
    @property
    def basis_index(self):
        if self._basis_index is not None:
            return self._basis_index
        return range(len(self.data))

    @basis_index.setter
    def basis_index(self, value):

        self._basis_index = value
        if hasattr(self, '_basis_distances'):
            del self._basis_distances

    @property
    def number_basis(self):
        return len(self.basis_index)

    @property
    def tree(self):
        if not hasattr(self, '_tree'):
            self._tree = BallTree(self.data, self.leaf_size)
        return self._tree

    @property
    def basis_distances(self):
        if not hasattr(self, '_basis_distances'):
            if self.number_basis > 3:
                basisPoints = self.data[self.basis_index]
                self._basis_tree = BallTree(basisPoints, leaf_size=3)
                self._basis_distances = np.array([np.mean(self._basis_tree.query(
                                     basisPt, k=3, return_distance=True
                                     )[0]) for basisPt in basisPoints])
            else:
                self._basis_distances = np.ones(self.basis_index)
        return self._basis_distances

    @tree.setter
    def tree(self, value):
        self._tree = value

    def query(self, double epsilon, basis=None, int k=3):
        """Query Ball Tree
        basis : index of basis functions to query for neighbors
        epsilon : size of nearest neighbor ball
        k : minimum nearest neighbors
        """
        cdef:
            int num_basis, i
        basis_dist = self.basis_distances

        if basis is not None:
            assert basis <= self.number_basis
            b = self.basis_index[[basis]]
            num_basis = b.size
            basis_dist = np.array([basis_dist[basis]])
        else:
            b = self.basis_index
            num_basis = self.number_basis
        radius = epsilon * basis_dist
        radius = np.tile(epsilon, num_basis)
        indices = [self.tree.query_radius(basis[np.newaxis, :], r=r,
                  return_distance=False)[0] for basis, r in izip(
                                             self.data[b, :], radius)]
        for i in xrange(num_basis):
            if len(indices[i]) < k:
                indices[i] = self.tree.query(self.data[self.basis_index[i]],
                                             k=k, return_distance=False)[0]
        return indices

    def __getstate__(self):
        d = {}
        d['data'] = self.data
        d['_basis_index'] = self._basis_index
        d['basis_distances'] = self.basis_distances
        if hasattr(self, '_tree'):
            d['_tree'] = self._tree
        return d

    def __setstate__(self, d):
        self.data = d['data']
        self._basis_index = d['_basis_index']
        if '_tree' in d:
            self._tree = d['_tree']
```

```
#!python
#cython: cdivision=True
#cython: boundscheck=False
#cython: wraparound=False
#cython: profile=True
#cython: embedsignature=True
import multiprocessing
from cython.parallel import prange, parallel, threadid
from libc.stdlib cimport abort, malloc, free
cimport openmp
from cython.view cimport array
import numpy as np
cimport numpy as np
cimport cython

DTYPE = np.float64
ctypedef np.float64_t DTYPE_t
ITYPE = np.int32
ctypedef np.int32_t ITYPE_t
np.import_array()
cdef int num_threads

@cython.boundscheck(False)
@cython.wraparound(False)
@cython.initializedcheck(False)
cdef api void mahalanobis_cdist(
                    double [:,::1] X1,
                    double [:,::1] X2,
                    double [:,:,::1] VI,
                    double [:,::1] Y):
    """Mahalanobis Distance
    Y=(X1-X2).T*VI*(X1-X2)
    """
    cdef:
        unsigned int i1, i2, m1, m2, n
        double [:,:, ::1] tmp
        double [:,:,::1] q
        double res
        unsigned int i, j, p1=0, p2=0, inc1=1, inc2=1, threads=1, thread_id

    m1 = X1.shape[0]
    m2 = X2.shape[0]
    d = VI.shape[1]
    n = X1.shape[1]
    assert d == VI.shape[2]
    assert d == X1.shape[1]
    assert d == X2.shape[1]
    assert m1 == Y.shape[0]
    assert m2 == Y.shape[1]

    if p2 > 60:
        num_threads = 6
    else:
        num_threads = 2

    q = np.zeros((num_threads, n, m1), dtype=DTYPE, order='c')
    tmp = np.zeros((num_threads, n, m1), dtype=DTYPE, order='c')

    for p2 in prange(m2, nogil=True, num_threads=num_threads, schedule='dynamic'):

        thread_id = threadid()
        tmp[thread_id, :, :] = 0
        for p1 in xrange(m1):
            for i in xrange(n):
                q[thread_id, i, p1] = -X2[p2, i]
        for i in xrange(n):
            matrix_add(X1[:, i], q[thread_id, i, :], 1)
        matrix_mult(VI[p2], q[thread_id], tmp[thread_id])
        for p1 in xrange(m1):
            res = 0.0
            for i in xrange(n):
                res = res + q[thread_id, i, p1] * tmp[thread_id, i, p1]
            Y[p1, p2] = res

cdef void matrix_add(double [:] A, double [:] B, double alpha) nogil:
    cdef:
        ITYPE_t N, incA, incB
    N = A.shape[0] #num rows in A

    incA = A.strides[0] // sizeof(double)
    incB = B.strides[0] // sizeof(double)

    cblas.daxpy(N, alpha, &A[0], incA, &B[0], incB)


cdef inline void matrix_mult(double [:, ::1] A, double [:, ::1] B, double [:, ::1] C) nogil:
    cdef:
        int M, N, K, incA, incB, incC
    M = A.shape[0] #num rows in A and C
    N = B.shape[1] #num col in B and C ##actually taking transpose of B in blas
    K = A.shape[1] #num cols in A also num rows in B

    incA = A.strides[0]//sizeof(double)
    incB = B.strides[0]//sizeof(double)
    incC = C.strides[0]//sizeof(double)

    cblas.dgemm(cblas.CblasRowMajor, cblas.CblasNoTrans, cblas.CblasNoTrans, M, N, K,
            1, &A[0,0], K, &B[0,0], N,
            0, &C[0,0], N)


def mahalanobis2_blas(np.ndarray[DTYPE_t, ndim=2] x1, np.ndarray[DTYPE_t, ndim=2] x2,
                      np.ndarray[DTYPE_t, ndim=3] Vi):
    '''Mahalanobis**2 Distance'''
    cdef:
        int m1, m2
        np.ndarray[DTYPE_t, ndim=2] Y

    m1 = x1.shape[0]
    m2 = x2.shape[0]

    x1 = np.asarray(x1, dtype=DTYPE, order='c')
    x2 = np.asarray(x2, dtype=DTYPE, order='c')
    Y = np.zeros((m1, m2), dtype=DTYPE, order='c')

    mahalanobis_cdist(x1, x2, Vi, Y)
    return Y
```

REFERENCES

[1] "Airforce technology website, http://www.airforce-technology.com/projects/jsf/."

[2] "Marine traffic, http://www.marinetraffic.com/ais."

[3] AARTS, E. H. L., ANDERSON, E. J., GENDREAU, M., GLASS, C. A., HERTZ, A., HONKALA, I. S., JOHNSON, D. S., KINDERVATER, G. A. P., KORST, J. H. M., LAARHOVEN, P. J. M. V., LAPORTE, G., and LENSTRA, J. K., *Local Search in combinatorial Optimization*. John Wiley & Sons Ltd, 1997.

[4] AGARD, B. and KUSIAK, A., "Standardization of components, products and processess with data mining," in *International Conference on Production Research Americas*, (Santiago, Chile), 2004.

[5] AHN, C. W. and RAMAKRISHNA, R. S., "Multiobjective real-coded bayesian optimization algorithm revisited: Diversity preservation," in *Genetic and Evolutionary Computation Conference*, pp. 593–600, July 7-11 2007.

[6] ANANDKUMAR, A., HSU, D., ADEL JAVAMARD, and KAKADE, S., "Learning linear bayesian networks with latent variables," in *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[7] ANDERBERG, M., "Cluster analysis for applications," tech. rep., OFFICE OF THE ASSISTANT FOR STUDY SUPPORT KIRTLAND AFB N MEX, 1973.

[8] ANDERSON, D., ZARE, A., and PRICE, S., "Comparing fuzzy, probabilistic and possibilistic partitions using the earth mover's distanc," *IEEE Transactions on Fuzzy Systems*, 2012.

[9] ARCHAMBEAU, C. and MICHEL VARLEYSEN, "Manifold constrained finite gaussian mixtures," *Computational Intelligence and Bioinspired Systems*, pp. 820–828, 2005.

[10] BAYES, T., "An essay towards solving a problem in the doctrine of chances," *Phil. Trans. of the Royal Soc. of London*, vol. 53, pp. 370–418, 1763.

[11] BENDIX, F., KOSARA, R., and HAUSER, H., "Parallel sets: Visual analysis of categorical data," *IEEE Symposium on Information Visualization (InfoVis)*, pp. 133–140, 2005.

[12] BENGIO, Y., LAROCHELLE, H., and VINCENT, P., "Non-local manifold parzen windows," *Advances in Neural Information Processing Systems*, pp. 115–122, 2005.

[13] BEZDEK, J. C. and HARRIS, J. D., "Fuzzy partitions and relations; an axiomatic basis for clustering," *Fuzzy*, vol. 1, pp. 111–127, 1978.

[14] BILTGEN, P., *Capability-Based Technology Evaluation for Systems-of-Systems*. PhD thesis, Georgia Institute of Technology, 2007.

[15] BILTGEN, P., ENDER, T., and MAVRIS, D., "Development of a collaborative capability-based tradeoff environment for complex systems architectures," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006.

[16] BISHOP, C., "Mixture density networks," tech. rep., Neural Computing Research Group, 1994.

[17] BISHOP, C. M., *Pattern Recognition and Machine Learning*. Springer, 2006.

[18] BROX, T., ROSENHAHN, B., CREMERS, D., and SEIDEL, H.-P., "Nonparametric density estimation with adaptive, anisotropic kernels for human motion tracking," in *Proceedings of the 2nd conference on Human motion: understanding, modeling, capture and animation*, pp. 152–165, 2001.

[19] BUTLER, A., "JSF tests slips again, purchase to be slashed," *Aviation Week*, January 2011.

[20] CHAMBERLAIN, M. K., *An approach to Decision Support for Strategic Redesign*. PhD thesis, Georgia Institute of Technology, 2007.

[21] CHANG, K. C. and TIAN, Z., "Efficient inference for mixed bayesian networks," in *Proceedings of the 5th ISIF/IEEE International Conference on Information Fusion*, 2002.

[22] CHASE, N., RADEMACHE, M., GOODMAN, E., AVERILL, R., and SIDHU, R., "A benchmark study of multi-objective optimization methods," tech. rep., Red Cedar Technology.

[23] CHEN, C. and WANG, L., "Product platform design through clustering analysis and information theoretical approach," *International Journal of Production Research*, vol. 46, pp. 4259–4284, 2008.

[24] CHENG, J. and DRUZDZEL, M. J., "Adaptive importance sampling in bayesian networks," *Journal on Artificial Intelligence*, vol. 13, pp. 155–188, 2000.

[25] CHICKERING, D. M., "Learning bayesian networks is np-complete," in *Learning from Data: Artificial Intelligence and Statistics V* (FISHER, D. and LENZ, H. J., eds.), 1996.

[26] CHICKERING, D., "Learning equivalence classes of bayesian network structures," in *Proceedings of the Twelfth Annual Conference on Unvertainty in Artificial Intelligence*, pp. 150–157, 1996.

[27] CHIN, H. and COOPER, G., "Bayesian beliefy network inference using simulation," *Uncertainty in Artificial Intelligence*, vol. 3, pp. 129–147, 1989.

[28] CHOWDHURY, S., MESSAC, A., and KHIRE, R., "Comprehensive product platform planning (cp3) framework: Presenting a generalized product family model," in *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structual Dynamics, and Materials Conference*, 2010.

[29] COOPER, G. and HERSKOVITS, E., "A bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[30] CRUZ-RAMIREZ, N., ACOSTA-MESA, H. G., BARRIENTOS-MARTNEZ, R. E., and NAVA-FERNANDEZ, L. A., "How good are the bayesian information criterion and the minimum description length principle for selection? a bayesian network analysis," in *Proceedings of the Fifth Mexican International Conference on Artificial Intelligence*, 2006.

[31] DA SAILVERIA, G., BORENSTEIN, D., and FOGLIATTO, F. S., "Mass customization: Literature review and research directions," *Internation Journal of Production Economics*, vol. 72, pp. 1–13, 2001.

[32] DABBEERU, M. M., DEB, K., and MUKERJEE, A., *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, ch. Product Portfolio Selection of Designs Through an Analysis of Lower-Dimensional Manifolds and Identification of Common Properties, pp. 161–187. Springer, 2011.

[33] DAI, Z. and SCOTT, M. J., "Product platform design through sensitivity analysis and cluster analysis," *Journal of Intellegent Manufacturing*, vol. 18, pp. 97–113, 2007.

[34] DALY, R., SHEN, Q., and AITKEN, S., "Learning bayesian networks: Approaches and issues," *The Knowledge Engineering Review*, vol. 26, no. 2, pp. 99–157, 2011.

[35] DAVIS, S., "From future perfect: mass customizing," *Planning Review*, vol. 17, pp. 16–21, 1989.

[36] DE OLIVEIRA, J. V. and PEDRYCZ, W., eds., *Advances in Fuzzy Clustering*. Wiley, 2007.

[37] DEB, K., PRATAP, A., AGARWAL, S., and MEYARIVAN, T., "A fast elitist multiobjective genetic algorithm nsaga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 187–197, 2002.

[38] DEMPSTER, A. P., LAIRD, N. M., and RUBIN, D. B., "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[39] DEUTCH, J. M., WIDNALL, S. E., and DALTON, J. H., "Charter for the joint advanced strike technology (jast) program," August 1994.

[40] ELIDAN, G., "Lighting-speed structure learning of nonlinear continuous networks," in *Proceedings of the 15th International Conference on Artificial Inteligence and Statistics*, 2012.

[41] FASSHAUER, G., "Positive definite kernels: Past, present and future," *Dolomite Research Notes on Approximation*, vol. 4, pp. 21–63, 2011.

[42] FAUGERAS, O. P., "A quantile-copula approach to conditional density estimation," *Journal of Multivariate Analysis*, vol. 100, no. 9, pp. 2083–2099, 2009.

[43] FELLINI, R., KIM, H. M., KOKKOLARAS, M., MICHELENA, N., and PAPALAMBROS, P., "Target for design of product families," in *Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization*, June 4-8 2001.

[44] FERGUSON, S., *Aerial Obervation of Oil Pollution at Sea*. Cedre, english translation ed., 2006. The Centre of Documentation, Research and Experimentation on Accidental Water Pollution.

[45] FERGUSON, S., KASPRZAK, E., and LEWIS, K., "Designing a family of reconfigurable vehicles using multilevel multidisciplinary design optimization," *Structural and Multidisciplinary Optimization*, vol. 39, pp. 171–186, 2009.

[46] FERGUSON, S., SIDDIQI, A., LEWIS, K., and DE WECK, O. L., "Flexible and reconfigurable systems: Nomenclature and review," *Proceedings of the ASME 2007 International Design Engineering Technical Conferences and Computers Information in Engineering Conference*, 2007.

[47] FISHER, M., RAMDAS, K., and ULRICH, K., "Component sharing in the management of product veriety: A study of automotive braking systems," *Management Science*, vol. 45, no. 3, pp. 297–315, 1999.

[48] FOULLOY, L. and BENOIT, E., "Building a class of fuzzy equivalence realations," *Fuzzy Sets and Systems*, vol. 157, pp. 1417–1437, 2006.

[49] FRASCA, M. and LIBERATI, R., "Riemann manifolds from hellinger distnace," *Tyrrhenian Workshop on Advances in Radar and Remote Sensing (TyWRRS)*, 2012.

[50] FRENO, A. and TRENTIN, E., *Hydrid Random Fields*. Spring, 2011.

[51] FUJITA, K., "Product variety optimization under modular architecture," *Computer Aided Design*, vol. 34, pp. 953–965, 2002.

[52] FUJITA, K. and YOSHIDA, H., “Product variety optimization: simultaneous optimization of module combination and module attributes,” in *ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, (Pittsburgh, PA), 2001.

[53] FUNG, R. and DEL FAVERO, B., “Backward simulation in bayesian networks,” in *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 102–109, 1994.

[54] FUNG, R. and CHANG, K.-C., “Weighting and integrating evidence for stochastic simulation in bayesian networks,” *Uncertainty in Artificial Intelligence*, vol. 5, pp. 209–219, 1989.

[55] GALLAGHER, M., WOOD, I., KEITH, J., and SOFRONOV, G., “Bayesian inference in estimation of distribution algorithms,” in *Proceedings IEEE Congress on Evolutionary Computation*, pp. 127–133, 2007.

[56] GLOBALSECURITY, “F-35 joint strike fighter (JSF) commonality,” *www.GlobalSecurity.org*.

[57] GONZALEZ-ZUGASTI, J., OTTO, K., and BAKER, J., “Assessing value in platformed product family design,” *Research in Engineering Design - Thoery, Applications, and Concurrent Engineering*, vol. 13, pp. 30–41, 2001.

[58] GUO, H. and HSU, W., “A survey of algorithms for real-time bayesian network inference,” *AAAI Technical Report WS-02-15*, 2002.

[59] GYFTODIMOS, E. and FLACH, P. A., “Hierarchical bayesian networks: an approach to classification and learning for structured data,” *Methods and Applications of Artificial Intelligence*, vol. Springer Berlin Heidelberg, pp. 291–300, 2004.

[60] HAUBELT, C. and RICHTER, K., “System design for flexibility,” *In Proceedings of Design, Automation and Test in Europe*, 2002.

[61] HAZELRIGG, G. A., “On the role and use of methematical models in engineering design,” *Journal of M*, vol. 131, pp. 336–341, 1999.

[62] HEATH, C. and GRAY, J., “Openmdao: Framework for flexible multidisciplinary design, analysis and optimization methods,” in *8th AIAA Multidisciplinary Design Optimization Specialist Conference (MDO)*, pp. 1–13, 2012.

[63] HENRION, M., “Propagating uncertainty in bayesian networks by probabilistic logic sampling,” *Uncertainty in Artificial Intelligence*, vol. 3, pp. 161–173, 1988.

[64] HERNANDEZ, G., *Platform Design for Customizable Products as a Problem of Access in Geometric Space*. PhD thesis, G.W. Woodruff School of Mechanical Engineering Georgia Institute of Technology, 2001.

[65] HERNANDEZ, G., ALLEN, J. K., and MISTREE, F., “Design of hierarchic platforms for customizable products,” in *Proceedings of DETC’02 ASME Design Automation Conference*, 2002.

[66] HOLMBERG, G., *On aircraft development- managing flexible complex systems with long life cycles*. PhD thesis, LinkÃ¶ping University, 2003.

[67] HOLMES, C., “Effective platform designs for medium lift helicopters,” Master’s thesis, Massachusetts Institute of Technology, 1999.

[68] HOLTTA, K., TANG, V., and SEERING, W., “Modularizing product architectures using dendrograms,” 2003.

[69] HOU, X., NI, X. S., and SMITH, A. K., *Mining of Enterprise Data*, ch. A Survey of Manifold-Based Learning Methods. 2007.

[70] ICKSTADT, K., BORNKAMP, B., GRZEGORCZYK, M., WIECZOREK, J., SHERIFF, M. R., GRECCO, K., and ZAMIR, E., “Nonparametric bayesian networks,” in *BAYESIAN STATISTICS* (BERNARDO, J., BAYARRI, M., BERGER, J., DAVID, A., HECKERMAN, D., SMITH, A., and WEST, M., eds.), vol. 9, 2010.

[71] INSELBERY, A., “The plane with parallel coordinates,” *The Visual Computer*, vol. 1, pp. 69–91, 1985.

[72] JIAO, J., SIMPSON, T. W., and SIDDIQUE, Z., "Product family design and platform-based product development: a state-of-the-art review," *Journal of Intellegent Manufacturing*, 2007.

[73] JIAO, J., ZHANG, Y., and WANG, Y., "A generic genetic algorithm for product family design," *Journal of Intellegent Manufacturing*, vol. 18, pp. 233–247, 2007.

[74] JONES, E., OLIPHANT, T., and OTHERS, "Scipy: Open source scientific tools for python,"

[75] JR, K. H. K., *Design Space Exploration of Stochastic System-Of-Systems Simulations Using Adaptive Sequential Experiments.* PhD thesis, Georgia Institute of Techgnology, 2012.

[76] JUNGHANS, M., *Visualization of Hyperedges in Fixed Graph Layouts.* PhD thesis, Brandenbury University of Technology Cottbus, 2008.

[77] KANDEL, A., *Fuzzy Techniques in Pattern Recognition.* John Wiley & Sons, 1982.

[78] KAZEMZADEH, R., BEHZADIAN, M., AGHDASI, M., and ALBADVI, A., "Integration of marketing research techniques into house of quality and product family design," *International Journal of Advanced Manufacturing Technology*, vol. 41, pp. 1019–1033, 2009.

[79] KERN, S., MÜLLER, S., HANSEN, N., BÜCHE, D., OCENASEK, J., and KOUMOUTSAKOS, P., "Learning probability distributions in continuous evolutionary algorithms - a comparative review," *Natural Computing*, vol. 3, pp. 77–112, 2004.

[80] KESTNER, B., MARTIN, K., PERULLO, C., SCHUTTE, J., and MAVRIS, D., "Integrated system design using bayesian belief networks," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2013.

[81] KHAJAVIRAD, A. and MICHALEK, J. J., "A decomposed gradient-based approach for generalized platform selection and variant design in product family optimization," *Journal of Mechanical Design*, vol. 130, 2008.

[82] KHAJAVIRAD, A., MICHALEK, J. J., and SIMPSON, T. W., "An efficient decomposed multiobjective genetic algorithm for solving the joint product platform selection and product family design problem with generalized commonality," *Structural & Multidisciplinary Optimization*, vol. 39, no. 2, pp. 187–201, 2009.

[83] KHIRE, R., WANG, J., BAILEY, T., and SIMPSON, T., "Product family commonality selection through interactive visualization," *International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, pp. 1–11, 2008.

[84] KHIRE, R., MESSAC, A., and SIMPSON, T. W., "Selection-integrated optimization (sio) methodology for adaptive systems and product family optimization," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[85] KOENKER, R. and BASSETT, G., "Regression quantiles," *Econometrica*, vol. 46, pp. 33–50, 1978.

[86] KOKKOLARAS M, FELLINI R, K. H. M. N. P. P., "Extension of the target cascading formulation to the design of product families," *Structural Multidisciplenary optimization*, vol. 24, pp. 293–301, 2002.

[87] KOLLER, D. and FRIEDMAN, N., *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[88] KRISHNAN, V. and GRUPTA, S., "Appropriateness and impact of platform based product development," *Management Science*, vol. 47, no. 1, pp. 52–68, 2001.

[89] KUMAR, D., CHEN, W., and SIMPSON, T. W., "A market-driven approach to the deisgn of platform-based product families," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[90] KUMAR, M. and COHEN, K., "Wild land fire fighting using multiple uninhabited aerial vehicles," in *AIAA Infotech@Aerospace Conference*, April 2009.

[91] LAMOTHE, J., HADJ-HAMOU, K., and ALDANONDO, M., "An optimization model for selecting a product family and designing its supply chain," *European Journal of Operational Research*, vol. 169, 2006.

[92] LEE, C., *Bayesian Collaborative Sampling: Adaptive Learning for Multidisplinary Design*. PhD thesis, Georgia Institute of Technology, 2011.

[93] LEVINA, E. and BICKEL, P., "The earth mover's distance is the mallows distance: Some insights from statistics," in *International Conference on Computer Vision*, pp. 251–256, 2001.

[94] LIU, J. S., *Monte Carlo Strategies in Scientific Computing*. New York: Springer Verlag, 2001.

[95] LOMHEIM, T., MILNE, E., KWOK, J., and TSUDA, A., "Performance sizing relationships for a short-wave mid-wave infrared scanning point-source detection space sensor," in *Aerospace Conference*, pp. 113–138, 1999.

[96] LZENMAN, A. J., *Modern Multivariate Statistical Techniques Regression, Classification, and Manifold Learning*. Springer, 2008.

[97] MARSHAL, A., "The use of multi-stage sampling schemes in monte carlo computations," in *Symposium on Monte Carlo Methods* (MEYER, M., ed.), pp. 123–140, 1956.

[98] MESSAC, A., MARTINEZ, M. P., and SIMPSON, T. W., "Effective product family design using physical programming," *Engineering Optimization*, vol. 34, no. 3, pp. 245–261, 2002.

[99] MESSAC, A., MARTINEZ, M. P., and SIMPSON, T. W., "A penalty function for product family design using physical programming," *ASME Journal of Mechanical Design*, vol. 124, no. 2, pp. 164–172, 2002.

[100] MEYER, M. and UTTERBACK, J., "The product family and the dynamics of core capability," *Sloan Management Review*, pp. 29–47, Spring 1993.

[101] MEYER, M. H. and LEHNERD, A. P., *The power of product platform - building value and cost leadship*. New York: Free Press, 1997.

[102] MICHELENA, N., KIM, H. M., and PAPALAMBROS, P., "A system partitioning and optimization approach to target cascading," *International Conference on Engineering Design*, 1999.

[103] MILLER, T. D. and ELGAARD, P., "Structuring principles for the designer," *International Design Seminar: Integration of Process Knowledge into Design Support Systems*, 1999.

[104] MOON, S. K., KUMARA, S. R., and SIMPSON, T. W., "Data mining and fuzzy clustering to support product family design," in *Proceedings of IDETC/CIE 2006*, 2006.

[105] MOSTELLER, F. and TUKEY, J., *Data Analysis and Regression*. Addison-Wesley Publishing Company, 1977.

[106] MURPHY, K. P., *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[107] NAYAK, R. U., CHEN, W., and SIMPSON, T. W., "A variation-based method for product family design," *Engineering Optimization*, vol. 34, no. 1, pp. 65–81, 2002.

[108] NEAPOLITAN, R., *Learning Bayesian Networks*. Pearson Prentice Hall, 2004.

[109] NELSON, S., PARKINSON, M., and PAPALAMBROS, P., "Multicriteria optimization in product platform design," *Journal of Mechanical Design*, vol. 123, no. 2, pp. 199–204, 2001.

[110] NEWCOMB, P. J., BRAS, B., and ROSEN, D. W., "Implications of modularity on product design for the life cycle," *Journal of Mechanical Design*, vol. 120, pp. 483–490, 1998.

[111] NIST/SEMATECH, "e-handbook of statistical methods," *Online at http://www.itl.nist.gov/div898/handbook/*, 2006. Updated July 18, 2006.

[112] OLEWNIK, A., BRAUEN, T., FERGUSON, S., and LEWIS, K., "A framework for flexible systems and its implementation in multiattribute decision making," *ASME Journal of Mechanical Design*, vol. 126, pp. 412–419, 2004.

[113] OLIPHANT, T., *Guide to numpy*. http://www.tramy.us/numpybook.pdf, Dec 2006.

[114] OLIVANDER, J., TARKIAN, M., and FENG, X., "Multi-objective optimization of a familly of industrial robots," in *Multi-Objective Evolutionary Optimization for Product Design and Manufacturing* (WANG, L., NG, A. H. C., and DEB, K., eds.), Springer, 2011.

[115] PATE, D. J., PATTERSON, M. D., and GERMAN, B. J., "Methods for optimizing a family of reconfigurable aircraft," *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, including the AIA*, September 2011.

[116] PAVLAKIS, P. and TARCHI, D., “On the monitorying of illicit vessel dischares using spaceborne sar remote sensing-a reconnaissance study in the mediterranean sea,” *Annals of Telecommunications*, pp. 700–718, 2001.

[117] PEARL, J., “Bayesian networks: A model of self-activated memory for evidential reasoning,” in *Proceedings of 7th Conference of the Cognitive Science Society*, 1985.

[118] PEARL, J., *Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference*. Morgan Kaulmann Publishing, Inc, 1988.

[119] PEDERSEN, K., *Designing platform families: an evolutionary approach to developing engineering systems*. PhD thesis, Georgia Institute of Technology, 1999.

[120] PELIKAN, M., *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. Springer, March 2005.

[121] PIRMORADI, Z. and WANG, G. G., “Recent advancements in product family design and platform-based product development: A literature review,” in *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2011.

[122] RAI, R. and ALLADA, V., “Modular product family design: agent-based pareto-optimization and quality loss function-based post-optimal analysis,” *Internation Journal of Production Research*, vol. 41, pp. 2075–4098, 2003.

[123] RAJAGOPALAN, B., LALL, U., TARBOTON, D. G., and BOWLES, D. S., “Multivariate nonparametric resampling scheme for generation of daily weather variables,” *Stochastic Hydrology and Hydraulics*, vol. 11, pp. 65–93, 1997.

[124] RAMIREZ, N. C., *Building Bayesian Networks from Data: a Constraint-based Approach*. PhD thesis, The University of Sheffield, 2001.

[125] RANQUE, P., FREEMAN, D., KERNSTINE, K., LIM, D., GARCIA, E., and MAVRIS, D., “Stochastic agent-based analysis of uav mission effectiveness,” in *11th AIAA ATIO Conference*, 2011.

[126] REICHERT, P., SCHERVISH, M., and SMALL, M., “An efficient sampling technique for bayesian inference with computationally demanding models,” *American Statistical Association and the American Society for Quality*, vol. 44, pp. 318–327, 2002.

[127] ROBERT, C. P., “Simulation of truncated normal variables,” *Statistics and Computing*, vol. 5, pp. 121–125, 1995.

[128] ROBERTSON, D. and ULRICH, K., “Planning for product platforms,” *Sloan Manage*, vol. 39, pp. 19–31, 1998.

[129] ROBINSON, R. W., “Counting labeled acyclic digraphs,” in *New Directions in the Theory of Graphs* (HARARY, F., ed.), p. 2390272, New York: Academic Press, 1973.

[130] ROGERS, A. and PRUGEL-BENNETT, A., “Genetic drift in genetic algorithm selection schemes,” *IEEE Transactions of Evolutionary Computation*, 1999.

[131] RUBNER, Y., TOMASI, C., and GUIBAS, L., “The earth mover’s distance as a metric for image retrieval,” *International Journal of Computer Vision*, vol. 40, pp. 99–121, 2000.

[132] SALEH, J. H., HASTINGS, D., and NEWMAN, D., “Flexibility in system design and implications for aerospace systems,” *Acta Astronautica*, vol. 53, pp. 927–944, 2003.

[133] SANTINI, S. and FELLOW, R., “Similarity measures,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999.

[134] SCHRAGE, D., “Technology for rotorcraft affordability through inintegrated product/process development (ippd),” in *Presented at the American Helicopter Society 55th Annual Forum*, (Montreal, Canada), May 25-29 1999.

[135] SCOTT, M. J. and ANTONSSON, E. K., “Aggregation functions for engineering design tradeoffs,” *Fuzzy Sets and Systems*, vol. 99, no. 3, pp. 253–264, 1998.

[136] SCOTT, S., LESH, N., and KLAU, G., “Investigating human-computer optimization,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2002.

[137] SHACHTER, R. and PEOT, M., "Simulation approaches to general probabilistic inference on belief networks," *Uncertainty in Artificial Intelligence*, vol. 5, pp. 221–231, 1989.

[138] SHAH, A. and WOOLF, P., "Python environment for bayesian learning: Inferring the structure of bayesian networks from knowledge and data," *Journal of Machine Learning Research*, vol. 10, pp. 159–162, 2009.

[139] SHAH, R., REED, P., and SIMPSON, T., "Many-objective evolutionary optimisation and visual analytics for product family design," in *Multi-Objective Evolutionary Optimization for Product Design and Manufacturing* (WANG, L., NG, A. H. C., and DEB, K., eds.), Springer, 2011.

[140] SHARMA, A., TARBOTON, D. G., and LALL, U., "Steamflow simulation: A nonparametric approach," *Water Resources Research*, vol. 33, pp. 291–308, 1997.

[141] SHAWE-TAYLOR, J. and CRISTIANINI, N., *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[142] SICHANI, P. K., *Separating Product Family Design Optimization Problems*. PhD thesis, University of Maryland, 2010.

[143] SIDDIQUE, Z. and ROSEN, D. W., "On combinatorial design spaces for the configuration design of product families," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 15, pp. 91–108, 2001.

[144] SIMPSON, T., "Methods for optimizing product platforms and product families: overview and classification," in *Product platform and product family design: methods and applications* (SIMPSON, T., SIDDIQUE, Z., and JIAO, J., eds.), pp. 133–156, Springer, 2005.

[145] SIMPSON, T. W., PEPLINSKI, J. D., KOCH, P. N., and ALLEN, J. K., "Metamodels for computer-based engineering design: Survey and recommendations," *Engineering with Computers*, vol. 17, pp. 129–150, 2001.

[146] SIMPSON, T., LIN, D., and CHEN, W., “Sampling strategies for computer experiments: Design and analysis,” *International Journal of Reliability and Application*, vol. 2, pp. 209–240, 2001.

[147] SIMPSON, T., MAIER, J., and MISTREE, F., “Product platform design: method and ap,” *Research Engineering Design*, vol. 13, pp. 2–22, 2001.

[148] SIMPSON, T. W., *A concept exploration method for product family design*. PhD thesis, Georgia Institute of Technology, 1998.

[149] SIMPSON, T. W. and D’SOUZA, B., “Assessing variable levels of platform commonality within a product family using a multiobjective genetic algorithm,” *Concurrent Engineering-Research and Applications*, vol. 12, no. 2, pp. 119–129, 2004.

[150] SIVARD, G., *A Generic Information Platform for Product Families*. PhD thesis, Royal Institute of Technology, 2000.

[151] SLINGERLAND, L. A., “A product family optimization approach using multidimensional data visualization,” Master’s thesis, Pennsylvania State University, 2010.

[152] STEDLE, C. E., “The joint strike fighter program,” *Johns Hopkins APL Technical Digest*, vol. 18, 1997.

[153] STONE, R. B., WOOD, K. L., and CRAWFORD, R. H., “A heuristic method for identifying modules for product architectures,” *Design Studies*, vol. 21(5), pp. 5–31, 2000.

[154] SUGIUAMA, M., TAKEUCHI, I., SUZUKI, T., KANAMORI, T., HACHIYA, H., and OKANOHARA, D., “Least-squares conditional density estimation,” *IEICE Transaction on Information and Systems*, vol. E93-D, pp. 583–594, 2010.

[155] SUH, N. P., *Axiomatic Design: Advances and Applications*. Oxford University Press, 2001.

[156] SULLIVAN, M., *Joint Strike Fighter Restructuring Should Improve Outcomes, but Progress Is Still Lagging Overall*. United States Government Accountability Office, March 2011.

[157] Takeuchi, I., Le, Q., Sears, T., and Smola, A., "Nonparametric quantile estimation," *Journal of Machine Learning Research*, vol. 7, pp. 1231–1264, 2006.

[158] Teh, Y. W. and Jordan, M., "Hierarchical bayesian nonparametric models with applications," in *Bayesian Nonparametrics: Principles and Practice*, pp. 158–207, Cambridge University Press, 2010.

[159] Theodoridis, S. and Koutroumbas, K., *Pattern Recognition 3rd Edition*. Academic Press, 2006.

[160] Tresp, V., "Mixtures of gaussian processes," *Advances in Neural Information Processing Systems*, vol. 13, pp. 654–660, 2001.

[161] Ulrich, K., "The role of product architecture in the manufacturing firm," *Research Policy*, vol. 24, pp. 419–440, 1995.

[162] Ulrich, K. and Eppinger, S., *Product Design and Development*. McGraw-Hill, 2nd ed. ed., 2000.

[163] Umeda, Y., Kondoh, S., Shimomura, Y., and Tomiyama, T., "Development of design methodology for upgradable products based on function-behavior-state modeling," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 19, pp. 161–182, 2005.

[164] Vincent, P. and Bengio, Y., "Manifold parzen windows," *Advances in Neural Information Processing Systems*, pp. 825–832, 2002.

[165] Vogel, K., Riggelsen, C., Nuehan, N., and Scherbaum, F., "Graphical models as surrogates for complex ground motion models," in *EGU General Assembly*, 2012.

[166] Wang, G. G. and Shan, S., "Review of metamodeling techniques in support of engineering design optimization," *Journal of Mechanical Design*, 2006.

[167] Wei, X., Wang, W., zhijie Li, and Liu, X., "Hypergraph model of product family structre for mass customization," in *International Conference on Management of e-Commerce and e-Government*, 2010.

[168] WILLIAMS, C., ALLEN, J., ROSEN, D., and MISTREE, F., “Designing platforms for customizable products and processes in markets of nonuniform demand,” *Concurrent Engineering*, vol. 15, pp. 201–216, 2007.

[169] YUAN, C., *Importance Sampling for Bayesian Networks: Principles, Algorithms, and Performance*. PhD thesis, University of Pittsburgh, 2006.

[170] ZHANG, Y., JIAO, J., and MA, Y., “Market segmentation for product family positioning based on fuzzy clustering,” *Journal of Engineering Design*, vol. 18, no. 3, pp. 227–241, 2007.

[171] ZHAO, Y., ZHANG, M., SU, N., and CHEN, J., “Product family extension configuration design: The theory and method,” *2nd International Conference on Computer and Automation Engineering (ICCAE)*, pp. 321–326, 2010.