ABSTRACT

Title of dissertation: DATA-DRIVEN GEOMETRIC DESIGN SPACE EXPLORATION AND DESIGN SYNTHESIS

Wei Chen, Doctor of Philosophy, 2019

Dissertation directed by: Professor Mark D. Fuge
Department of Mechanical Engineering

A design space is the space of all potential design candidates. While the design space can be of any kind, this work focuses on exploring geometric design spaces, where geometric parameters are used to represent designs and will largely affect a given design's functionality or performance (*e.g.*, airfoil, hull, and car body designs). By exploring the design space, we evaluate different design choices and look for desired solutions. However, a design space may have unnecessarily high dimensionality and implicit boundaries, which makes it difficult to explore. Also, if we synthesize new designs by randomly sampling design variables in the high-dimensional design space, there is a high chance that the designs are not feasible, as correlation exists between feasible design variables. This dissertation introduces ways of capturing a compact representation (which we call a *latent space*) that describes the variability of designs, so that we can synthesize designs and explore design options using this compact representation instead of the original high-dimensional design variables. The main research question answered by this dissertation is: how does one effectively learn this compact representation from data and efficiently explore this latent space so that we

can quickly find desired design solutions? The word "quickly" here means to eliminate or reduce the iterative ideation, prototyping, and evaluation steps in a conventional design process. This also reduces human intervention, and hence facilitates design automation.

This work bridges the gap between machine learning and geometric design in engineering. It contributes new pieces of knowledge within two topics: design space exploration and design synthesis. Specifically, the main contributions are:

1. A method for measuring the intrinsic complexity of a design space based on design data manifolds.

2. Machine learning models that incorporate prior knowledge from the domain of design to improve latent space exploration and design synthesis quality.

3. New design space exploration tools that expand the design space and search for desired designs in an unbounded space.

4. Geometrical design space benchmarks with controllable complexity for testing data-driven design space exploration and design synthesis.

DATA-DRIVEN GEOMETRIC DESIGN SPACE EXPLORATION AND DESIGN SYNTHESIS

by

Wei Chen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor Mark D. Fuge, Chair/Advisor
Professor Shapour Azarm
Professor Katrina Groth
Professor David Jacobs (Dean's Representative)
Professor Linda Schmidt

Acknowledgments

Over the past four years of my PhD study, I have received tremendous help and support from people around me. Dr. Mark Fuge, who is the most brilliant, responsible, and considerate advisor I have ever met, has been consistently offering guidance on not only research, but also my career path. You gave me great advice for particular problems, while also kept reminding me to step back and see the bigger picture. Also because of you, I was able to work efficiently in a nice and friendly environment in the past four years. I would also like to thank my committee members, Shapour Azarm, Katrina Groth, David Jacobs, and Linda Schmidt, who have provided insightful comments for my research.

In addition, I would like to acknowledge my colleagues from the IDEAL lab at UMD. I have learned a lot from our collaborations and discussions. Also thanks to the Product Design, Modeling and Simulation Research Group in Siemens Corporate Technology. You offered me a great internship experience—an opportunity where I can integrate my research into real products.

Finally, I am grateful that my family and friends are always providing me the strongest support. You have brought me happiness, helped me go through hard times, and given me the motivation to excel in my research.

Table of Contents

Chapter 1: Introduction

A *design space* is the space of all potential design candidates defined by *design variables*. Through exploring the design space, we can look for desired solutions by evaluating different design alternatives before realizing them in the real world. While some of the methods described in this dissertation can be applied to any design space, the primary focus is on geometric design spaces. *Geometric design* in general is a branch of computational geometry and focuses on curve and surface modeling and representation. Here, by geometric design, we refer to the design of any object's geometry. For example, it can be the design of the car body, mechanical parts, or shape of any object, as opposed to the design of materials, color, or machining processes. A geometric design can have various representations. For example, we can represent a two-dimensional design by using spline curves [105, 179] or grid-point coordinates [40, 49], while a three-dimensional design can be represented by free-form deformation (FFD) [66] or point clouds [228]. Although different representations lead to different design spaces, geometric design spaces are generally high-dimensional and complex, which complicates both human exploration and computational optimization. This dissertation addresses this by analyzing geometric design spaces and generating new representations that are more interpretable and compact.

1.1 Why use Data-Driven Methods to Create Design Representations?

Many traditional design processes go through the cycle of ideation, prototyping, evaluating, and improving. This iterative process usually requires human intervention, which can be expensive and time-consuming. Eventually, only a few designs are implemented and tested, leaving limited time for exploring design alternatives. Designers can find a design that satisfies all requirements (*i.e.*, feasible to all constraints), but the true optimal design is hard to uncover since insufficient design alternatives are explored.

Alternatively, one can represent a class of designs by a consistent parameterization, so that a design can be synthesized by setting those parameters and the process of searching for solutions can be automated by optimization algorithms. For example, people have parameterized designs using polynomials, splines, or free-form deformation (FFD) to represent aerodynamic or hydrodynamic designs (*e.g.*, airfoils and hulls) [66, 105, 122, 163, 179]. Given a parameterization, designs can be optimized in that parameter space (*i.e.*, the design space) by using methods like the genetic algorithm [79] or topology optimization [16, 140, 147]. These types of approaches also have downsides:

1. The dimensionality of the design space is often higher than the minimal dimensions needed to capture design variability, which creates unnecessary computational cost. For example, if we use B-spline curves to parameterize airfoils, the design space will be the coordinates of those B-spline control points. However, we cannot arbitrarily change these coordinates (*e.g.*, because airfoils possess properties like curvature continuity, *etc.*). Thus feasible designs lie on a *low-dimensional manifold* within that large design space. The dimensionality of this manifold is the minimal dimensionality that preserves design variability.

We call it the *intrinsic dimensionality*.

2. Designs that can be represented by a certain parameterization are usually limited. This can be mitigated by adding parameters (*e.g.*, increasing the number of B-spline control points). However, it is hard to decide on the number of parameters (*i.e.*, design space dimensionality) manually, as we need to trade-off between representation capacity and computational cost.

3. The boundaries of the design space are often unknown. It is also tricky to set these boundaries, because the design space should be large enough to include the optimal solution, while as small as possible to reduce the computational cost on exploring infeasible regions.

To address some of these downsides, we can extract useful information from existing designs and leverage this information to create new designs. Designers have made many designs based on certain knowledge set. Usually, new designs are created based on the same knowledge set. So why can't we make machines learn the knowledge set from previous designs, and infer new designs based on the learned knowledge set? Techniques from machine learning have made this possible. Once trained properly, machine learning models can automatically capture the intrinsic dimensionality, variability, and boundaries of designs from data, thus ameliorating the above problems. In this dissertation, I propose data-driven methods to address problems in traditional design, and facilitate the design process so that it frees the time of designers, provides inspiration for them, and even allows practitioners to create products without the help of a design expert.

1.2 Research Questions and Contributions of the Dissertation

Data-driven design, especially in the field of engineering, usually requires extra consideration compared to problems dealt with in common machine learning areas like computer vision and language processing. Specifically, the following three aspects make data-driven design difficult:

1. Unlike the bounded input space of images or sentences, the bounds for the design space are usually unknown, so one has to assume the bounds during tasks like design optimization. For example, people usually predefine the parameter ranges of given parametric models when optimizing airfoils or hulls [105, 122, 163]. As mentioned previously, these predefined design space boundaries are sometimes problematic.

2. Design synthesis differs from synthesis commonly studied in the machine learning scenario (*e.g.*, image, music, sentence, or video synthesis), as it is subject to stricter functional (*e.g.*, the lift/drag coefficient of airfoils) or geometrical (*e.g.*, surface smoothness) constraints.

3. There are special requirements associated with data-driven design. For example, we want to regularize the latent space such that designs change consistently along any direction in the latent space. This property is desirable for design exploration over the latent space, as it regularizes the mapping from the latent space to the design space or the performance space.

In response to these difficulties, this dissertation explores the following fundamental research questions and contributions needed to answer those questions:

RQ 1: How does one measure the complexity of a design space? **Contribution:**

Chapter 3 represents this complexity by the intrinsic dimensionality, non-linearity, and separability of the design data manifold [40, 49].

RQ 2: How does one find a low-dimensional latent representation that captures design variability? **Contribution:** Chapter 3 derives latent spaces by using different models and then evaluates each latent space. Chapter 5 further describes a model that hierarchically captures a regularized latent space of each component in a design [50].

RQ 3: How does one explore the design space without specifying a fixed boundary? **Contribution:** Chapters 6 and 7 describe an adaptive sampling method that identifies feasible regions and a Bayesian optimization method that searches for global optimal designs, respectively, without the need for predefined design space bounds [43, 44]. This also allows the discovery of novel designs outside the domain of existing designs.

By building data-driven models to find out answers to these questions, we can solve the three problems in traditional design described earlier. Particularly, the proposed data-driven design methods will help in three ways:

1. Feasible domain identification. By estimating a function that predicts the feasibility of designs, we can identify the feasible domains of the design space with respect to implicit constraints.

2. Design space exploration. We can capture a *latent space* that represents the low-dimensional manifold encompassing the major variability of designs. Design space exploration can then be performed on this latent space instead of the higher-dimensional design space.

3. Design recommendation. We can generate new designs from the learned model that captures real-world design variability. These new designs will provide inspiration for designers and allow practitioners to create products without the help of design experts.

1.3 Overview

In this dissertation, I will use design data to solve existing problems in design, and help facilitate design space exploration and design synthesis. Specifically, each chapter addresses the following:

Chapter 2: reviews previous work that helps data-driven design space exploration and synthesis.

Chapter 3: shows how to measure the intrinsic complexity and dimensionality of a design space, and generate fundamental knowledge about how designs differ from one another. It deepens our understanding of design complexity in general.

Chapter 4: introduces a deep generative model of aerodynamic designs (specifically airfoils) that reduces the dimensionality of the optimization problem by learning from shape variations in an airfoil database, and proposes a two-stage optimization method that prioritizes the optimization of major attributes.

Chapter 5: describes a method to synthesize designs with inter-part dependencies. It decomposes the design space of the whole design into the design space of each component while keeping the inter-component dependencies satisfied, so that we can perform design synthesis and design space exploration for each component separately.

Chapter 6: introduces Active Expansion Sampling (AES), a method that identifies (possibly disconnected) feasible domains over an unbounded input space. It avoids the need for setting explicit design variable bounds, and can be used for discovering potentially novel and creative designs.

Chapter 7: proposes a Bayesian optimization approach, Trust Region Bayesian Optimization (TRBO), that only needs to specify an initial search space that does not necessarily include the global optimum, and expands the search space when necessary.

While the primary focus of this dissertation is on geometric design spaces, some of the methods described here can be applied to other design spaces. Specifically, when demonstrating dimensionality reduction techniques in Chapters 3-5, although we only use geometrical designs as examples since they are easy to visualize, the proposed techniques are applicable to studying any design spaces that can be reduced and represented by a lower-dimensional space. The design space exploration methods described in Chapters 6 and 7 can be used in any design space.

1.4 How to Use This Dissertation

Although this dissertation is organized in a way that each chapter should be read sequentially, specific audiences may benefit more from certain portions of the work.

Engineering Design Practitioners looking to develop data-driven design automation applications. Chapters 4 and 5 discuss the technique for learning generative models from design databases, so that new designs can be automatically synthesized from those models. Sections 4.5-4.6 and Chapters 6-7 further introduce

tools that can be built on those generative models and automatically search for desired designs.

Machine Learning Researchers looking for new domains of application. Despite the wide usage of machine learning in areas including social media, robotics, and medical science, its application in engineering design is currently limited. In general, the whole dissertation discusses machine learning techniques that address engineering design problems. Particularly, Chapter 2 reviews how we can use dimensionality reduction and Bayesian approaches to facilitate design synthesis and design space exploration. Sections 3.4.1 and 5.4.1 provide synthetic datasets that could serve as benchmarks for design space study. Finally, Section 8.3 brings up some open questions and future research directions that may be addressed by more advanced machine learning techniques.

Those who are interested in design of experiments, system design, algorithm hyperparameter optimization, structural optimization, *etc.* Chapters 6 and 7 present design space exploration tools based on the Bayesian approach. Specifically, Chapter 6 introduces an adaptive sampling method for feasible domain identification and Chapter 7 introduces a global optimization method. Different from conventional methods, they gradually expand the search space while updating the Bayesian model, so that the operating domain is no longer limited within fixed variable bounds. They are most useful when the variable ranges are hard to specify.

Researchers looking to reproduce the results of this dissertation. The code and data for reproducing the experimental results are open-source through the GitHub link in each chapter.

Chapter 2: Background

This dissertation focuses on the data-driven solution of two tasks: design space exploration and design synthesis. Both tasks have been dealt with in areas of design, machine learning, and computer graphics. This chapter reviews some fundamental concepts and the existing state of the art in data-driven design space exploration and synthesis.

2.1 Geometric Design Representation

To create a geometric design space, one has to define a consistent representation that can express a set of designs. Note that in most cases the representation has to be consistent (*e.g.*, having the same dimensionality[1] and one-to-one mapping from a representation to a design) to form a design space, which allows machine learning or optimization algorithms to operate on.

Using discrete geometry is a straight-forward way to represent designs. Specifically, sequences of point coordinates is used for representing 2D shapes that consist of curves or pen strokes [40, 49, 91]. For 3D models, we use representations such as point clouds [71, 99, 228], depth maps [203], geometry images [196, 197], voxel grids [24, 137, 199, 231], and octrees [176, 212, 226]. These discrete representations

[1]Note that sometimes a design can also be represented as a sequence of variable length (*i.e.*, variable dimensionality) [91]. In such cases we may still optimize the design as long as we can get a consistent latent representation.

are usually flexible enough to model any possible shape variation. However, it is their flexibility that makes those representations under-constrained. For example, when generating 3D models with smooth surfaces, the representation of voxel grids cannot enforce the smoothness by itself. Another issue is that they often suffer from their high dimensionality, which is also the source of their representation flexibility.

In contrast, using parameterization of shapes as a representation reduces the dimensionality of discrete representations and usually enforces properties like smoothness and watertightness. The spline curve is a common 2D shape parameterization approach, and is used commonly for aerodynamic shapes [105, 179]. Representation such as abstract deformation handles [241, 242] and free-form deformation (FFD) [66] are used for 3D models. Though these representations enforce desirable properties, they also limit the diversity of shapes so that some minor features may not be represented. Also, the mapping from a parameterization and a design is sometimes not one-to-one. For example, different rational Bézier curve representations (*i.e.*, control points, weights and parameter variables combinations) can result in the same curve. This may bring difficulties to exploring the design space.

2.2 The Manifold Hypothesis

Though parameterization reduces the design space dimensionality required for discrete geometry representations, the intrinsic dimensionality is usually much lower. Thus a large body of work has been focusing on reducing the design space to a low-dimensional latent space.

The design space is where potential design candidates exist. We can find desired designs by searching the design space using methods such as the genetic algorithm [131]. However, the design space contains mostly invalid solutions, and valid

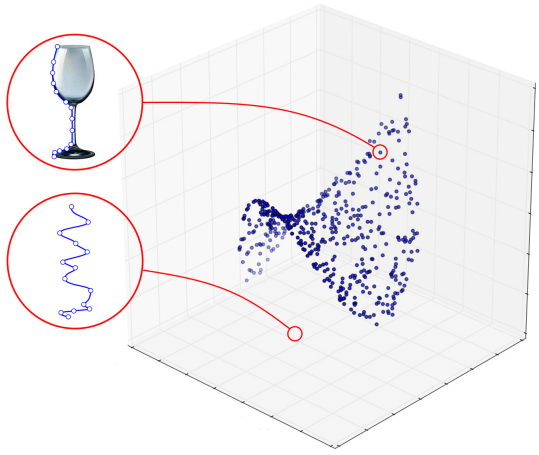


Figure 2.1: Valid glass designs (parameterized by Bézier curves) are on a low-dimensional manifold. Here the high-dimensional design space is shown in 3D for visualization.

designs are usually on a lower-dimensional manifold (Fig. 2.1). Also, based on the curse of dimensionality [14], the cost of exploring the design space grows exponentially with its dimensionality. It is expensive and unnecessary to directly search a high-dimensional geometric design space $\mathcal{X}$. Instead, we can capture one or multiple lower-dimensional manifolds (*i.e.*, the latent space $\mathcal{Z}$) that valid designs lie on. This is done by learning a mapping $f : \mathcal{X} \to \mathcal{Z}$. Since the latent space captures major design variability, we can then search for designs in the latent space.

2.2.1 Learning the Latent Space

Since the high-dimensional design space is hard to visualize or explore and contains mostly invalid solutions, we can use a lower-dimensional latent space to capture the variability of designs. Previous efforts have been made to achieve the goal of discovering how designs vary in the latent space. Work in this area is highly related to dimensionality reduction, manifold learning, and representation learning.

We can model the mapping $f : \mathcal{X} \to \mathcal{Z}$ as a linear mapping, which uses a set of

optimal directions or basis functions to represent designs such that the variance of shape geometry is maximized. Such methods include the Karhunen-Loève expansion (KLE) [51, 66], principal component analysis (PCA) [69], and the active subspaces approach [214]. These methods usually have closed-form solutions to both the mapping f and its reverse mapping g. However, in practice, it is more reasonable to assume that design variables lie on a non-linear manifold, rather than a hyperplane. This brings up the need to study non-linear methods. The non-linearity can be achieved by 1) applying linear reduction techniques locally to construct a non-linear global manifold [69, 130, 167–169]; 2) using kernel methods with linear reduction techniques [49, 69]; 3) latent variable models like Gaussian process latent variable model (GPLVM) and generative topographic mapping (GTM) [224]; and 4) neural networks based approaches such as self-organizing maps [165] and autoencoders [27, 49, 59, 69].

These previous approaches are for capturing a low-dimensional latent space that represents major variability of designs. They are usually applied for reducing the complexity of design optimization [165, 224] or creating interactive tools to visualize and explore the design space [8, 162, 241].

2.2.2 Exploring the Latent Space

When designs are associated with requirements or performance scores, we can find feasible or optimal designs by relating the latent space to relevant quantities of interest (QoI).

In applications like design space exploration [64, 128, 237] and reliability analysis [133, 248], people need to find feasible domains within which solutions are valid. Usually these constraints are implicit and cannot be expressed analytically (*e.g.*, aesthetics, functionality, or performance requirements) and require expensive evalu-

ations (*e.g.*, simulation, experiments, or human evaluation). This problem is studied in the fields of adaptive sampling and active learning, where we want the mapping h from the design space $\mathcal{X}$ to the evaluation space $\mathcal{Y} \in \{-1, 1\}$ (-1 for infeasible and 1 for feasible designs) to be estimated through as few measurements as possible. Specifically, we iteratively update the estimated mapping h. Within each iteration t, a design $\boldsymbol{x}^{(t)}$ is chosen to be evaluated such that its result $y^{(t)} \in \mathcal{Y}$ is the most informative for updating the estimation—this property is referred to as *informativeness* within the Active Learning community. This informativeness can be based on label ambiguity [100, 136, 188], estimated expected error [31, 125, 154, 247], the reduction of the *version space* [215], classifier disagreement [7, 148], or the predictive variance [26, 86, 111].

Similar approaches have also been developed for design optimization, where the evaluation space $\mathcal{Y} \in \mathbb{R}$ in most cases. Bayesian optimization (BO) is one type of method that aims to reduce the number of evaluations. Instead of selecting the most informative design at each iteration, BO selects designs to evaluate based on their likelihood of being the optimal design. This likelihood is quantified as probability of improvement [127], expected improvement [217], or upper confidence bounds (UCB) [206].

When using adaptive sampling to identify feasible domains or using Bayesian optimization to find global optimal designs, we normally need to specify some fixed design variable bounds. In cases such as algorithm hyperparameter tuning [190, 204, 207] and shape optimization [164], setting the variable bounds are not trivial. It is hard to guarantee that any fixed bounds will include the entire feasible domain or the true global optimum. To solve this problem, this dissertation demonstrates new methods for adaptive sampling or Bayesian optimization that require no fixed design variable bounds. This also allows the discovery of novel designs outside the domain

of existing designs.

Due to the curse of dimensionality, the number of evaluations required to precisely estimate h increases exponentially with the dimensionality of $\mathcal{X}$. Thus instead of learning the mapping $h : \mathcal{X} \to \mathcal{Y}$, we can learn a different mapping $h' : \mathcal{Z} \to \mathcal{Y}$, given that we have the mapping from the latent space to the design space, *i.e.*, $g : \mathcal{Z} \to \mathcal{X}$. We need g for synthesizing designs to perform the evaluation (*i.e.*, experiments or simulations) and get the desired results (feasible or optimal designs). This brings up the need for design synthesis.

2.3 Designs Synthesis

Design synthesis is not only used in feasible domain identification and design optimization, but also for simplifying the design process by recommending new designs and visualizing the variability of design candidates. While this dissertation mainly deals with data-driven design synthesis, there are two other approaches to generating new designs, namely rule-based approaches and assembly-based approaches.

2.3.1 Rule-Based Approaches

Rule-based approaches generate new shapes or designs via explicit rules or constraints. These approaches involves procedural modeling [150, 210] and computational design synthesis (CDS) [30, 118, 119, 158, 232]. However, such explicit rules are usually hard-coded based on prior knowledge, and are difficult to specify or generalize to diverse design objectives.

Instead of requiring rules of shapes themselves, topology optimization [16, 140, 147] specifies a set of rules including a performance objective, constraints, and the design's physical interaction with the environment (*i.e.*, loads and/or boundary condi-

tions), which are usually clear in a specific problem definition. Topology optimization generates new designs by searching for the solution of an optimization problem defined by those rules. Since normally topology optimization relies on gradient-based optimization, it cannot be applied on black-box objectives (*e.g.*, aesthetics, customer preference, or experimental performance).

2.3.2 Assembly-Based Approaches

Assembly-based approaches synthesize new shapes by assembling or reorganizing parts from an existing shape database, while preserving the desired structures [36, 109, 208, 234, 246]. The shapes are usually parameterized by high-level abstract representations, such as hand-crafted feature vectors [109] or shape grammars [208]. While these methods generate designs by using parts in the database, they edit shapes at a high-level and do not control each part's local geometry.

2.3.3 How Data-Driven Approaches Differ from Other Synthesis Approaches

In contrast to rule-based or assembly-based design synthesis, data-driven approaches learn rules, constraints, and shape deformation from data. They eliminate the need for explicit rules, and often generate designs with more flexibility. More importantly, we can identify a continuous compact latent space that encodes major variations of valid designs. This simplifies design space exploration.

As mentioned previously, we embed designs into low-dimensional latent spaces by learning the mapping $f : \mathcal{X} \to \mathcal{Z}$. Meanwhile, given any coordinate in a latent space, one can also synthesize a design if the inverse mapping $g : \mathcal{Z} \to \mathcal{X}$ is also learned. Usually, dimensionality reduction techniques allow inverse transformations

from the latent space back to the design space, thus can synthesize new designs from latent variables [49, 51, 59, 69]. For example, under the PCA model, the latent variables define a linear combination of principal components to synthesize a new design [51]; for local manifold based approaches, a new design can be synthesized via interpolation between neighboring points on the local manifold [130]; and under the autoencoder model, the trained decoder maps any given point in the latent space to a new design [27, 59, 197].

Different from models like PCA and autoencoders, a generative model learns the probability distribution of designs $P_{\boldsymbol{x}}$. Usually it projects latent variables $\boldsymbol{z}$ draw from some prior distribution $P_{\boldsymbol{z}}$ to a design $\boldsymbol{x} \sim P_{\boldsymbol{x}}$. Researchers have employed generative models such as kernel density estimation [209], Boltzmann machines [99], variational autoencoders (VAEs) [153], and generative adversarial nets (GANs) [137, 231] to learn the distribution of samples in the design space, and synthesize new designs by drawing samples from the learned distribution.

Though the aforementioned machine learning models can be directly applied to design synthesis, in practice there are limitations that hinder their performance. For example, engineering designs are usually subject to specific geometrical (*e.g.*, surface smoothness of aerodynamic shapes) considerations that subjects commonly synthesized in the machine learning scenario (*e.g.*, images, music, sentences, or videos) do not have. These considerations are prior knowledge that we can incorporate into our model to improve data-driven design synthesis.

To guide data-driven design space exploration and design synthesis, the next chapter starts by understanding important properties of a design space, *i.e.*, intrinsic dimension, data separability, and non-linearity.

Chapter 3: Learning the Intrinsic Complexity of Design Spaces

Portions of the work in this chapter were published in the Journal of Mechanical Design [49] and the ASME International Design Technical Conference [41]

3.1 Introduction

Products differ among many design parameters. For example, a wine glass contour can be designed using coordinates of B-spline control points: with 20 B-spline control points, a glass would have at least 40 design parameters. However, we cannot arbitrarily set these parameters because the contour must still look like a wine glass. Therefore, high-dimensional design parameters actually lie on a lower-dimensional *design manifold* (Fig. 3.1) or *latent space* that encodes major variability of design attributes, such as roundness or slenderness. A manifold's *intrinsic dimension* is the minimal dimensionality we need to faithfully represent how those high-dimensional design parameters vary.

Past researchers, both within and beyond design, have proposed many algorithms for mapping high-dimensional spaces to lower-dimensional manifolds and back again—what we call a *design embedding*. But how do you ensure the embedding has captured all the geometric variability among a collection of designs, while not using more dimensions than necessary? How do you evaluate which embedding best captures a given design space? What properties should a good design embedding

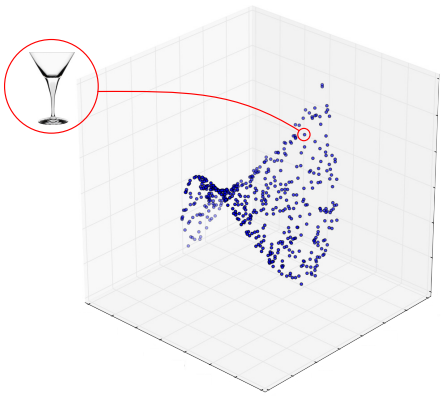


Figure 3.1: 3D visualization of high dimensional design space showing that design parameters actually lie on a 2-dimensional manifold.

possess?

This chapter answers those questions by proposing methods to study design embeddings. Specifically, our method measures the complexity of a high-dimensional design space and the quality of an embedding. We demonstrate our approach on synthetic *superformula* examples [79] with varying complexity and real-world glassware and airfoil examples.

While this chapter focuses on mathematically understanding design spaces, our approach ultimately has implications for several important sub-fields of Engineering Design. In engineering optimization, the number of design variables severely impacts accuracy and convergence. In consumer preference models, high-dimensional design spaces complicate capturing human opinion inexpensively or accurately. In design interfaces, designers have difficulty exploring and manipulating high-dimensional design spaces. Because our approach automatically determines a design space's complexity and dimension, these sub-fields can assess 1) their task's fundamental difficulty, and 2) how to best reduce that difficulty. Mathematically, our work deepens our understanding of design complexity in general by illuminating how design embeddings model a design space, how to judge the inherent complexity of a design space, and

how to measure the ways designs differ from one another.

Our main contributions are:

1. A method for embedding designs in the fewest dimensions needed to control shape variations.

2. A minimal set of performance metrics for embeddings, including reconstruction error, topology preservation, and whether the manifold captures key shape variations.

3. Manifold benchmarks with controllable complexity (*i.e.*, non-linearity, dimensionality, and separability) for testing design embeddings.

3.2 Related Work

Past research has taken two different approaches to understanding design spaces and synthesizing new shapes or designs—*knowledge-driven methods* and *data-driven methods* [233]. Knowledge-driven methods generate new shapes or designs via explicit rules. One representative example is procedural modeling, which creates 3D models and textures of objects (such as buildings and cities) from sets of rules [150, 210]. Another example is Computational Design Synthesis (CDS), which synthesizes designs (such as gearboxes and bicycle frames) based on topological or parametric rules or constraints [30, 118, 119, 158, 232]. However, such explicit rules are hard to specify or generalize to diverse design objectives. Data-driven methods, by contrast, learn the representation of geometric structure from examples, adjusting the model to match provided design data. We refer the readers to the survey by Xu *et al.* [233] for an overview of data-driven shape processing techniques.

Our work uses a data-driven approach to learn the inherent complexity and the latent representation of a shape collection, and synthesize shapes by exploring that

latent representation. Generally, there are three main data-driven approaches: 1) assembly-based modeling, where parts from an existing shape database are assembled onto a new shapes [35–37, 90, 109]; 2) statistical-based modeling, where a probability distribution is fitted to shapes in the design space, and plausible shapes are generated based on that distribution [73, 109, 209]; and 3) shape editing, where a low-dimensional representation is learned from high-dimensional design parameters, and designers create shapes by exploring that low-dimensional representation.

This chapter falls into that third category. Many approaches use manifold learning techniques such as Multi-Dimensional Scaling (MDS) to map the design space to a low-dimensional embedding space [8]. However such approaches generally construct only one-way mappings (high to low). Some methods can also directly learn a two-way mapping between the design space and the embedding space, such as autoencoders, which uses neural networks to project designs from high to low dimension and back again [28, 240]. Another way is to associate shapes with their *semantic attributes* by crowd-sourcing, and then learn a mapping from the semantic attributes to the shapes, such that new shapes can be generated by editing these semantic attributes [241].

3.3 How Our Contributions Relate to Prior Work

Our hypothesis is that while shapes are represented in a high-dimensional design space, they actually vary in a lower-dimensional manifold, or surface within the larger design space [17, 216]. One would then navigate a shape space by moving across this manifold—like an ant walking across a surface of a sphere—while still visiting all valid designs. This raises several questions: How does one find the manifold (if it exists)? How does one "jump back" to the high-dimensional space (raw geometry) once one has wandered around on the manifold? How does one choose a "useful" manifold,

and how do ones evaluate that "usefulness?"

A common issue for previous shape synthesis methods is that they often do not address the inherent properties (*e.g.*, intrinsic dimension, non-linearity) of the geometric design space before embedding, choosing instead to set parameters (such as dimensionality) manually. This causes problems during embedding and shape synthesis because this may not adequately capture shape variability or may use unnecessary dimensions along which designs do not vary. Unlike past work, this chapter directly investigates the inherent complexity of a design space under the assumption that different parts of that space may differ in complexity. We discover information such as discontinuities in the design space and the intrinsic dimension of each segment. We then adjust the embeddings and design manifolds based on that information.

Another issue is that given various embeddings (constructed by MDS, autoencoders, *etc.*), how does one choose the embedding that best enables a smooth and accurate exploration of the space? Reconstruction error is one common metric for evaluating embeddings. It measures how accurately the model can reconstruct input data as it embeds data from high dimension to low and back again. However, sometimes embeddings can excel at reconstruction but ultimately place the data in a lower-dimensional space with unexpected and unintuitive topological structures, such as linear filaments in Deep Autoencoders [68]. While these structures may aid accurate reconstruction, they make it difficult for users to explore the embedded space. In this chapter, we propose evaluation metrics that measure three properties of an embedding: (1) its reconstruction accuracy, (2) how well it preserves a design space's topology, and (3) whether it captures the design space's principal semantic attributes. In practice, there is often a trade-off between these metrics, and our approach allows a designer to visualize and decide among embeddings with respect to that trade-off.

3.4 Samples and Data

Before we discuss our approach, we need to introduce some concrete benchmarks of design spaces that we will demonstrate and validate our method over. By design space, we mean any M-dimensional vector ($x \in \mathbb{R}^M$) that controls a given design's form or function—*e.g.*, its shape, material, power, *etc.* To create a high-dimensional design space $\mathcal{X}$, we generate a set of design parameters or shape representations $X \in \mathcal{X}$. For ease of explanation and visualization, this chapter uses designs described by 2D curved contours; however, the proposed methods extend to non-geometric design spaces as well. Specifically, we uniformly sample points along the shape contours and use their coordinates as X where $x_j^{(i)}$ and $y_j^{(i)}$ are the x and y coordinates of the jth point on the contour of the ith sample:

$$X = \begin{bmatrix} x_1^{(1)} & y_1^{(1)} & \dots & x_n^{(1)} & y_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{(m)} & y_1^{(m)} & \dots & x_n^{(m)} & y_n^{(m)} \end{bmatrix}$$

The sample shapes come from two sources: 1) the *superformula* [79] as a synthetic example whose design space complexity we can directly control, and 2) glassware and airfoil contours as real-world examples.

3.4.1 Synthetic Benchmark

Since this chapter's goal is to capture a design space's inherent properties, we first need a benchmark dataset whose properties we can directly control; this allows us to measure performance with respect to a known ground truth. Since, to our knowledge, no such benchmark exists for design embeddings, we created one using a generalization of the ellipse—called the *superformula* (See Fig. 3.2 for examples)—that allows

Figure 3.2: Examples of superformula shapes.

us to control the following properties: (non-)Linearity, number of separate manifolds, intrinsic dimension, and manifold separability/intersection. Two-dimensional superformula shapes have a multidimensional parameter space in $\mathbb{R}^6$ with parameters (a, b, m, n_1, n_2, n_3) [79] in Eqn. (3.1). With a radius (r) and an angle (θ), the superformula is expressed in polar coordinates as:

$$r(\theta) = \left(\left| \frac{\cos(\frac{m\theta}{4})}{a} \right|^{n_2} + \left| \frac{\sin(\frac{m\theta}{4})}{b} \right|^{n_3} \right)^{-\frac{1}{n_1}} \tag{3.1}$$

thus the Cartesian coordinates X are:

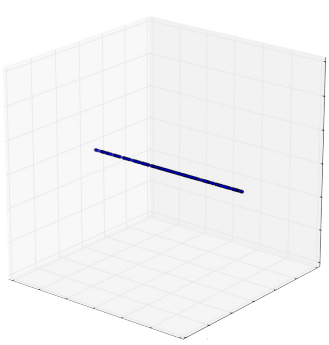
$$(x, y) = (r(\theta)\cos\theta, r(\theta)\sin\theta)$$

Controlling linearity. By tuning the parameters in Eq. 3.1 we can vary the non-linearity of X; for example, by changing the aspect ratio s of the shapes, we can linearly vary X:

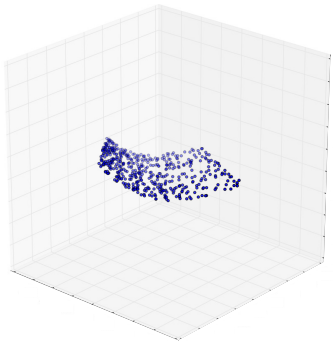
$$(x, y) = (s \cdot r(\theta)\cos\theta, r(\theta)\sin\theta) \tag{3.2}$$

We can control the linearity of X by tuning the linear switch s in Eqn. (3.2) or the nonlinear switches $\{a, b, m, n_1, n_2, n_3\}$ in Eqn. (3.1). Fig. 3.3a and 3.3b are examples of controlling linearity of the design space. A design space's linearity is reflected by the curvature of the manifold—higher non-linearity results in higher curvature.

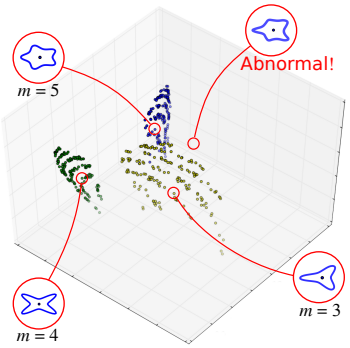
Controlling the intrinsic dimension. To artificially control the intrinsic dimension M of the design space, we construct M-dimensional subspaces of the superfor-



(a) Linear design space ($d = 1$) varying s.



(b) Nonlinear space ($d = 2$) varying s & n_3.



(c) Design space with multiple shape categories.

Figure 3.3: 3D visualization of the superformula design space created by a linear mapping from the high-dimensional design space $\mathcal{X}$ to a 3-dimensional space, solely for visualization. Each point represents a design.

mula parameter space by varying M parameters choosing from $\{s, a, b, m, n_1, n_2, n_3\}$, and keeping other parameters fixed, as shown in Fig. 3.3a (1-D) and 3.3b (2-D).

Controlling the number of shape categories. A collection of designs does not always consist of just one category of shapes. For example, a collection may contain not only glassware, but also bottles, which have very different contours to glassware and likely have different manifold properties. A naïve embedding, which lumps glasses and bottles together, should perform poorly here, and thus we need our synthetic benchmark to create similar cases with distinct manifolds. It is possible to have multiple categories of superformula shapes by discretely changing the value of m in Eqn. (3.1). Because the parameter m controls the period of the right-hand-side function in Eqn. (3.1), we can use it to set the superformula to a m-pointed-star, as shown in Fig. 3.3c. The discrete change of m forms separate clusters or sub-manifolds in the design space.

Another benefit of this formulation is that we can control the separability of the sub-manifold, since we can make these clusters intersect one another (*e.g.*, Fig. 3.7b). We can generate intersecting clusters by setting the ranges of varying parameters n_2 and n_3, such that all the clusters contain ellipses or circles. Because this superformula benchmark can generate design manifolds with many different properties, we believe it should be useful not only for benchmarking and improving future design embedding techniques, but also for evaluating manifold learning techniques in general.

3.4.2 Real-World Data

Glassware. Glassware is a good real-world example because 1) shape representation using B-splines smoothly fits the glass contours, and 2) we can interpret the semantic attributes of glassware—*e.g.*, roundness, slenderness, type of drink, *etc.* We use 128

glass images, including wine, beer, champagne, and cocktail glasses. We fit each glass contour with a B-spline, and build the design space $\mathcal{X}$ using the coordinates of points uniformly sampled across the B-spline curves.

Airfoils. An airfoil is the cross-section shape of a wing or blade (of a propeller, rotor, or turbine). Like glassware, we can also represent airfoils via 2D contours and they have discernible semantic attributes that allow us to verify different embeddings. A good airfoil shape embedding can aid airfoil optimization; for example, the proposed method can provide a continuous space with the fewest number of necessary dimensions for an airfoil optimization algorithm to optimize over, improving convergence speed and accuracy. Our airfoil samples are from the UIUC Airfoil Coordinates Database, which provides the Cartesian coordinates for nearly 1,600 airfoils.[1] We use linear interpolation to ensure that each airfoil has the same number of coordinates in the design space $\mathcal{X}$.

3.5 Methodology

First, we try to achieve good design embeddings by understanding the complexity or properties of the design space—for example, detecting the number of sub-manifolds and what their dimension might be. Based on those results, we then apply embeddings of appropriate complexity and type to the different sub-manifolds. We will review the details of our methodology. Interested readers can review the full source code for needed to reproduce all detail in this chapter.[2]

[1] http://m-selig.ae.illinois.edu/ads/coord_database.html
[2] https://github.com/IDEALLab/design_embeddings_jmd_2016

3.5.1 Pre-processing

Our raw data may come from shapes with various height and width, or have inconsistent or very high dimensionality. These issues obstruct manifold learning and embedding. We apply shape correspondence and linear dimension reduction as pre-processing steps to mitigate these issues.

Shape correspondence. We ensure that the coordinates for all designs correspond to consistent cardinality and areas in space. This step is important for our techniques to work well; however, the choice of correspondence technique is not central to the contributions of the chapter. In this work, we fixed the coordinates of the start and end points, and uniformly sample a constant number of points on the shape outline. Specifically,

1. For the superformula example, we set the point whose angle is 0 in the polar coordinate system as the start point, and the point which has the largest angle in the polar coordinate system as the end point;

2. For the glass example, we set the start and end point at the top and bottom of the left contour respectively;

3. For an airfoil, its outline has two curves — the upper and lower curves. We set the left and right most point as the start and end point of each curve, respectively.

Then for the superformula and glass examples, we standardized each sample such that the y coordinates are in [0, 1]; and for the airfoil example, we standardized each sample such that the x coordinates are in [0, 1].

Linear dimension reduction. Before performing non-linear dimension reduction, we first use Principal Component Analysis as a linear mapping ($f' : \mathcal{X} \in \mathbb{R}^{\mathrm{D}} \rightarrow \mathcal{X}' \in \mathbb{R}^{\mathrm{D}'}$) to reduce $\mathcal{X}$ such that it retains 99.5% of the variance. The original design space $\mathcal{X}$ is normally high dimensional. For example, if 100 points are used to represent the contour of each glass, we will have a 200-dimensional design space. It is difficult to learn from such a high dimensional space because of the following reasons: 1) the standard Euclidean distance metric is no longer a reliable measurement of similarity in a high dimensional space; 2) the number of samples required grows exponentially with the dimension; and 3) computing high dimensional data requires more time and resources. The first two reasons are the consequences of what is called the *curse of dimensionality* [15]. Despite their high dimensionality, the design parameters X are usually highly redundant. For example, in our experiments, by using principal component analysis (PCA), the 200-dimensional glassware design parameters can be reduced to 30-dimensional while still retaining at least 99.5% of the variance between designs. Therefore, our first step is to apply simple linear dimensionality reduction to map the high-dimensional design space to a lower (but still high) dimensional space:

$$f' : \mathcal{X} \in \mathbb{R}^{\mathrm{D}} \rightarrow \mathcal{X}' \in \mathbb{R}^{\mathrm{D}'}$$

where f' is a linear mapping and $D \geq D'$.

3.5.2 Learning Design Space Properties

If designs *do* lie on manifolds, we first need to know the number, intrinsic dimension, and complexity of those manifolds for two reasons. First, knowing the intrinsic dimension M sets the dimensionality d of the latent space $\mathcal{F}$. Setting $d < M$ makes it impossible to completely capture all variability of designs; while $d > M$ introduces

unnecessary dimensions along which designs do not vary—this unnecessarily impedes exploration or optimization.

Second, separating different categories of designs helps exclude invalid designs. To illustrate this point, we can first look into the case when there are multiple manifolds in a design space, as shown in Fig. 3.3c. Suppose we treat these multiple manifolds as one, and perform embedding and shape synthesis. We have to use a 3-dimensional latent space to completely capture the variation between the designs. Since there are no design samples in between two manifolds—what we call a *design cavity*, we do not know whether a design from that area is valid. Consequently, in that area we might synthesize a new shape which looks like a weird hybrid of two designs from two different manifolds, like the abnormal shape in Fig. 3.3c. In contrast, if we separate these manifolds and then do embedding and shape synthesis on each manifold/design category, we can avoid generating invalid new designs.[3]

For the above reasons, we apply clustering and intrinsic dimension estimation over the dimensionality-reduced design space $\mathcal{X}'$ before embedding designs.

Clustering. To separate manifolds, we use a method based on *robust multiple manifolds structure learning (RMMSL)* [83]. It assumes that within each cluster, samples should not only be close to each other, but also form a flat and smooth manifold. A manifold can be flat and smooth if it has small curvature everywhere. The method thus constructs an affinity matrix which incorporates both pairwise distance and a curvature measurement, which is computed via principal angles between local tangent spaces.

RMMSL uses a *curved-level* measurement $R(x)$ to indicate curvature and flatness:

[3] An astute reader may notice that designs "off-the-manifold" may be, in some sense, creative or innovative. We return to that discussion in Sect. 3.6.3.

$$R(x) = \sum_{x_i \in N(x)} \frac{\|\theta(J_i, J)\|}{d(x_i, x)} \tag{3.3}$$

where $\theta(J_i, J)$ measures the principal angle between the tangent spaces J_i at x_i and J at x, which indicates the curvature of the manifold; $d(x_i, x)$ is the geodesic distance between x_i and x; and $N(x)$ is the spatial neighborhood point set for x.

In general, RMMSL first performs local manifold structure estimation: it estimates the local tangent space $J_i \in \mathbb{R}^{\mathrm{D}' \times \mathrm{d_i}}$ at each sample x_i, where $i = 1, ..., n$ and d_i is the local intrinsic dimension at x_i (the method of estimating d_i is introduced in Sect. 5.2.2). Then it learns the global manifold structure by constructing an affinity matrix $W \in \mathbb{R}^{\mathrm{n} \times \mathrm{n}}$ using both a pairwise distance kernel $w_1(x_i, x_j)$ and a curved level kernel $w_2(x_i, x_j)$:

$$W_{ij} = w_1(x_i, x_j) w_2(x_i, x_j) \tag{3.4}$$

where $w_1(x_i, x_j) = exp(-\|x_i - x_j\|^2/(\sigma_i \sigma_j))$, $w_2(x_i, x_j) = exp(-\theta(J_i, J_j)^2/(\|x_i - x_j\|^2 \sigma_c^2/\sigma_i \sigma_j))$, σ_i and σ_j are local bandwidth [244], and σ_c is a coefficient to control the effect of the curved level kernel.

The time complexity for estimating the local tangent spaces is $O(ND'(k_{max}^2 + D'k_{max} + D'^2))$, where k_{max} is the maximum neighborhood size. The complexity for computing the affinity matrix using pairwise distances and curved-level measurements is $O(N^2 D' d_{max}^2)$, where d_{max} is the maximum local intrinsic dimension. Thus the overall computational cost for getting the affinity matrix is $O(ND'(k_{max}^2 + D'k_{max} + D'^2 + Nd_{max}^2))$.

To compute the curvature measurement, we need the neighboring point set $N(x)$ for each sample point x. We use the neighborhood contraction and expansion algorithm proposed by Zhang *et al.* [245]. The basic idea is that the neighborhood size k selected for each point x should not only reflect the local geometric structure of the

manifold, but also have enough overlap among nearby neighborhoods to allow local information propagation.

Specifically, the neighborhood contraction and expansion algorithm first uses k-NN to get a neighborhood $\mathcal{N}_i$ of size k_{max} for each point x_i, and constructs a matrix $X_i - \bar{x}_i 1^T$, where $X_i = [x_{i_1}, ..., x_{i_{k_{max}}}]$ and $\bar{x}_i = \sum_{k \leq k_{max}} x_{i_k}$. Then it contracts $\mathcal{N}_i$ by removing the farthest neighbor until some preset minimal neighborhood size k_{min} is reached or until the following inequality holds:

$$\sqrt{\sum_{j>d_i} (\sigma_j^{(i)})^2} \leq \eta \sqrt{\sum_{j \leq d_i} (\sigma_j^{(i)})^2} \tag{3.5}$$

where $\sigma_j^{(i)}$ is the jth singular value of $X_i - \bar{x}_i 1^T$ ($\sigma_1^{(i)} \geq ... \geq \sigma_{k_{max}}^{(i)}$), d_i is the local intrinsic dimension of the manifold at x_i, which we will introduce in the next section. and the small constant $\eta \in (0, 0.5)$.

Based on [245], for N samples the algorithm has complexity $O(N(k_{max} - k_{min})k_{max}^2(k_{max} + D'))$, when sample $x_i \in \mathbb{R}^{D'}$.

Normally given the correct number of manifolds C (*i.e.*, group number), RMMSL has good performances in separating them [83]. However, it requires manually specifying the number C. To automatically detect the group number C, we apply a method based on *self-tuning spectral clustering (STSC)* [244]. This method automatically infers C by exploiting the structure of the eigenvectors V of the normalized affinity matrix A (*i.e.*, the Laplacian matrix) using an iterative algorithm (with T number of iterations).

For each possible group number C, the STSC algorithm tries to find a rotation $\hat{R}$ such that each row in the matrix $Z = V_C \hat{R}$ has a single non-zero entry (V_C is the

first C columns of V). This is achieved by minimizing a cost function:

$$L = \sum_{i=1}^{n} \sum_{j=1}^{C} (Z_{ij}^2 - M_i^2)$$

where $M_i = \max_j Z_{ij}$. Then the best group number C is the one with the lowest cost. Once we get the optimal Z, the sample x_i can be assigned to cluster c if and only if $\max_j(Z_{ij}^2) = Z_{ic}^2$.

The eigenvalue decomposition of the Laplacian matrix takes $O(N^3)$ time. Given a group number C, at each iteration of the optimization, the time for computing Z via Givens rotation [82] is $O(C^2(C^3 + N))$. Thus the total time for computing Z is $O(TC_{max}^3(C_{max}^3 + N))$, where C_{max} is the maximum group number, and T is the number of iterations. Assigning samples to clusters takes $O(NC_{max})$ time. Therefore the overall computational cost for estimating the optimal group number is $O(N^3 + TC_{max}^3(C_{max}^3 + N))$. While the scale complexity on C_{max} is high, in practice C_{max} is often a small number, so the complexity penalty is manageable.

In sum, we first obtain the nearest neighbors $N(x_i)$ for each sample x_i, then apply RMMSL to compute the affinity matrix W. Finally, we use W as the affinity matrix for STSC to determine the group number C and assign samples to different groups.

Intrinsic dimension estimation. Following the manifold clustering procedure, we apply intrinsic dimension estimation over each manifold/design category. The estimator is based on the local dimension estimation method mentioned in RMMSL [83]. We first obtain the K nearest neighbors of each sample x_i. We set the neighborhood size K using the adaptive method proposed in [180]. With the neighbors of x_i and a weight matrix S, we construct a local structure matrix:

$$T_i = (X_i - x_i 1^T) S S^T (X_i - x_i 1^T)^T$$

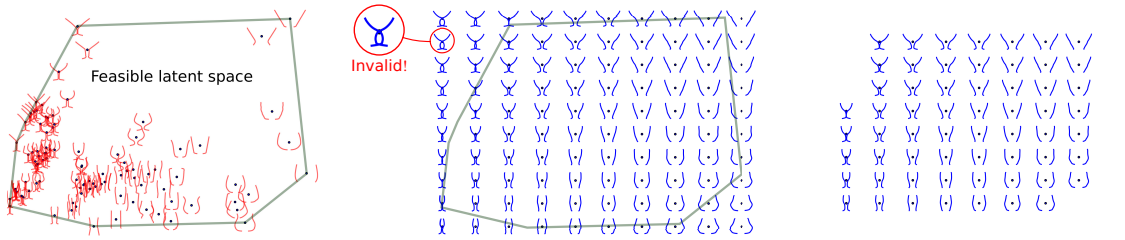
where S is a diagonal matrix and can be set as $S_{jj} = 1/(\sigma_n^2 + \sigma^2\alpha(\|x_{i_j} - x_i\|_2))$, σ_n^2 and σ^2 indicate the scales of the noise and the error, and $\alpha(\cdot)$ is a monotonically non-decreasing function with non-negative domain (*e.g.*, a quadratic) [83].

The local intrinsic dimension d_i is estimated from the eigenvalues of T_i—similar to dimensionality estimation using PCA. A category's overall intrinsic dimension is the mean intrinsic dimension of all points in that category. In practice, point-wise intrinsic dimension can be noisy (and thus appear to change dimensionality often). We apply a kernel density smoother to local dimensionality estimates to account for our assumption that the local dimensionality should not vary drastically within a neighborhood on a smooth manifold. For example, if the local dimensionality estimations for x_i and its five neighbors are $\{2, 3, 2, 2, 2, 2\}$, the second estimation is likely to be incorrect. After applying KDE, the new estimations will be $\{2, 2, 2, 2, 2, 2\}$. We use the Epanechnikov kernel to limit density estimation to a local neighborhood, and set the kernel bandwidth adaptively based on the distance between each sample and its Kth neighbor.

3.5.3 Embedding and Shape Synthesis

We used methods involving PCA, kernel PCA (with a RBF kernel) [183], and stacked denoising autoencoders (SdA) [221], which all simultaneously learn a mapping f from the space $\mathcal{X}'$ to the latent space $\mathcal{F}$ ($f : \mathcal{X}' \in \mathbb{R}^{\mathrm{D}'} \rightarrow \mathcal{F} \in \mathbb{R}^{\mathrm{d}}$) and a reverse mapping g from the $\mathcal{F}$ back to $\mathcal{X}'$ ($g : \mathcal{F} \in \mathbb{R}^{\mathrm{d}} \rightarrow \mathcal{X}' \in \mathbb{R}^{\mathrm{D}'}$) where $D' \geq d$.

PCA performs an orthogonal linear transformation on the input data. Kernel PCA extends PCA, achieving a nonlinear transformation via the kernel trick [182]. The SdA extends the stacked autoencoder [220], which is an multilayer artificial neural network that nonlinearly maps $\mathcal{X}'$ and $\mathcal{F}$ using nonlinear activation functions [18].



(a) Create a convex hull of the training set in the latent space.

(b) Copy the boundary of the convex hull to the grid of new designs generated from the latent space.

(c) Remove designs outside the boundary.

Figure 3.4: Set boundary of the feasible latent space.

We split any design data into a training set and test set. We further split the training data via 5-fold cross validation to optimize the hyperparameters using the *sequential model-based algorithm configuration (SMAC)* [102]. After optimizing any hyperparameters, the model trains each example with the entire training set. We test each model with the test set.

After mapping from design space $\mathcal{X}'$ to latent space $\mathcal{F}$, we also need to map back to the original $\mathcal{X}$ from $\mathcal{F}$—*i.e.*, given certain latent codes, we want to generate new shapes. This is achieved by first applying mapping g, and then applying g' ($g' : \mathcal{X}' \in \mathbb{R}^{\mathrm{D}'} \rightarrow \mathcal{X} \in \mathbb{R}^{\mathrm{D}}$), which is the inverse mapping of f'.

To visualize the mapping $g' : \mathcal{F} \rightarrow \mathcal{X}$, we uniformly sample from $\mathcal{F}$, map those samples back to $\mathcal{X}$ to synthesize their shapes, and then plot them in Fig. 3.4b. Note that although we only visualize a limited number of shapes generated from the latent space $\mathcal{F}$, we can generate infinitely many shapes when continuously exploring in $\mathcal{F}$.

Every latent space should have a boundary beyond which designs are not guaranteed to be valid. For example, at the top left of Fig. 3.4b, the glass contours on the two sides intersect. We call these *infeasible shapes*, *i.e.*, shapes that are unrealistic or invalid in the real-world. To limit $\mathcal{F}$ to only feasible shapes, we take the convex hull of the training samples in $\mathcal{F}$ as shown in Fig. 3.4a, and set its boundary as the

boundary for the *feasible latent space*. We sample and synthesize shapes inside the feasible latent space, as shown in Fig. 3.4c. Because training samples all have feasible shapes, any designs lying between any two training samples should be valid if the latent space preserves the original design space's topology (*i.e.*, the space is not highly distorted). As a result, this method may not explore innovative, unusual designs. While this chapter's main focus is understanding the complexity of an *existing* design space, we discuss how to find innovative designs in Sect. 3.6.3.

3.5.4 Evaluation

To evaluate embeddings, we consider these questions:

1. Given known latent codes (*e.g.*, roundness and slenderness), can the embedding precisely restore the original design parameters that created it?

2. Do shapes in the latent space ($\mathcal{F}$) vary similarly compared to the original shapes in the design space ($\mathcal{X}$)?

3. Does the embedding precisely capture all the attributes that control the variation of shapes?

To answer these questions, we propose three metrics for comparing embeddings: 1) reconstruction error, 2) geodesic distance inconsistency, and 3) principal attributes.

Reconstruction error. Reconstruction error measures how the actual design parameters X' differ from the design parameters of the input data once we project them onto the low-dimensional manifold and un-project back into high-dimensional space. We use the *symmetric mean absolute percentage error (SMAPE)* [144] to measure reconstruction error:

$$\epsilon = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{|r_j^{(i)} - s_j^{(i)}|}{|r_j^{(i)}| + |s_j^{(i)}|} \tag{3.6}$$

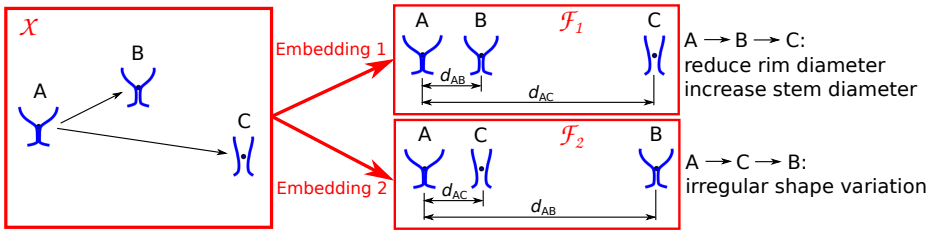


Figure 3.5: Illustration of pairwise distance preservation. Similar designs (A and B) have similar shape representations in $\mathcal{X}$, thus are closer in $\mathcal{X}$ than dissimilar designs (A and C). We want such relation of pairwise distances to be preserved in $\mathcal{F}$ (i.e., $d_{AB} < d_{AC}$) such that shapes will vary in the same manner as they do in $\mathcal{X}$.

where m is the sample size, n is the number of design parameters for each sample, $r_j^{(i)}$ is the ith reconstructed design parameter for the jth sample, and $s_j^{(i)}$ is the ith original design parameter for the jth sample.

Pairwise distance preservation. To answer the second question, we can compare the pairwise distances of samples in the high-dimensional design space $\mathcal{X}$ versus the low-dimensional latent space $\mathcal{F}$. Generally, as shown in Fig. 3.5, similar designs (A and B) have similar shape representations (e.g., Cartesian coordinates of shape outlines) in $\mathcal{X}$, thus are closer in $\mathcal{X}$ than dissimilar designs (A and C). We want such pairwise distances to be preserved in $\mathcal{F}$ (i.e., $d_{AB} < d_{AC}$) such that shapes will vary in the same manner as they do in $\mathcal{X}$. Since we assume our samples lie on a manifold in $\mathcal{X}$, we use the pairwise geodesic distances along the manifold as the pairwise distances to be preserved after the embedding.

Specifically, we construct a nearest neighbor graph G over the samples $X \in \mathcal{X}$ to model the manifold structure. G is a weighted graph where the edge weight between neighbors is their Euclidean distance. The nearest neighbors are selected using the neighborhood contraction and expansion algorithm mentioned in the neighborhood selection section. This algorithm adaptively chooses the neighborhood size for each sample based on its local manifold geometry. The neighborhood size is large where

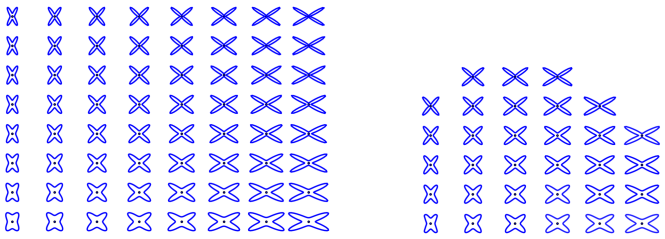
the manifold is flat, and small where it is curvy. Thus this method prevents "shortcuts" across high-curvature manifolds and maintains large enough overlap among nearby neighborhoods. Then we compute all pairs shortest paths for the graph G, and construct a geodesic distance matrix D_G. We compare D_G with the pairwise Euclidean distance matrix D of the embedded samples $F \in \mathcal{F}$ using Pearson's correlation coefficient. The *geodesic distance inconsistency (GDI)* can be expressed as

$$\text{GDI} = 1 - \rho(D_G, D)^2 \tag{3.7}$$

where $\rho(D_G, D)$ is the Pearson's correlation coefficient between D_G and D. Lower GDI indicates the embedding better preserves pairwise distances.

Normally, embedding methods like Isomap will have low GDIs because they explicitly optimize pairwise distances. However, they cannot simultaneously learn two-way mappings between the design space and the embedding space. Methods like autoencoders are able to learn two-way mappings, but they usually have higher GDIs because they minimize reconstruction error rather than preserve distances. There is often a trade-off between reconstruction and distance preservation; embeddings that explicitly optimize both objectives would be an interesting topic for future research.

Principal attributes. For superformula examples, we know what their correct latent spaces should look like by looking at their parameter spaces. A *parameter space* $\mathcal{P}$ (*e.g.*, Fig. 3.6a) for the superformula contains the shapes generated by all possible combinations of values for all the parameters used to modify the shapes. It is this space from which we randomly select training and testing samples. For example, if we fix parameters $\{a, b, m, n_2, n_3\}$ and vary $\{s, n_1\}$ in Eqn. (3.1) and Eqn. (3.2), the superformula parameter space will have two dimensions (*i.e.*, $\mathcal{P} \in \mathbb{R}^2$). Along the first dimension some shape attribute (*e.g.*, aspect ratio) changes with s, and along



(a) Shapes in the superformula parameter space.

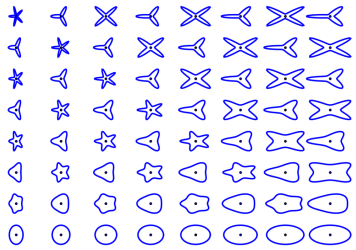
(b) Generated shapes in the latent space.

Figure 3.6: An example comparing shapes generated from the latent space $\mathcal{F}$ versus the superformula parameter space $\mathcal{P}$. If the embedding precisely captures the principal attributes, shapes from $\mathcal{F}$ should look like those from $\mathcal{P}$—with neither extra unexpected shape variation nor missing diversity.

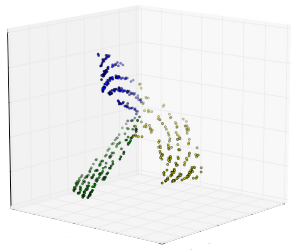
the second dimension another attribute (*e.g.*, roundness) changes with n_1. We call these attributes the *principal attributes*. We can also vary these two attributes along a single dimension by simultaneously varying s and n_1 (*e.g.*, let $s = \alpha t$ and $n_1 = \beta t$, where α and β are coefficients, and t is a variable). Fig. 3.7a shows an example where two shape attributes vary along each dimension—the aspect ratio[4] changes along one dimension and roundness along the other, and the number of arms changes over both dimensions. Because the number of arms is not continuous, the clustering algorithm should separate shapes with a different number of arms. And within each cluster, a good embedding should capture the other two attributes—the aspect ratio and roundness.

A good embedding should precisely capture the right principal attributes, such that tuning these attributes creates diverse but valid shapes. By comparing the shapes generated from the latent space $\mathcal{F}$ with those from the superformula parameter space $\mathcal{P}$, we can evaluate whether the embedding precisely captures all the attributes that control the variation of shapes. That is, shapes generated from $\mathcal{F}$ should look similar to those from $\mathcal{P}$—with neither unexpected shape variation nor missing diversity, as

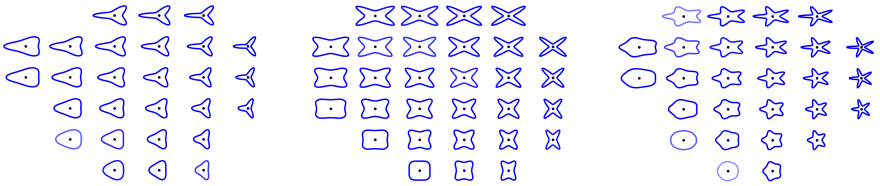
[4]All the shapes shown in the figures are scaled to the same height. Thus the aspect ratio will matter instead of the height or the width.



(a) Shapes in the superformula parameter space.



(b) Result of manifold clustering (as in Fig. 3.3, the design space $\mathcal{X}$ is visualized in three dimensions).



(c) Generated shapes in latent spaces. Since there are three categories, we have three separated latent spaces.

Figure 3.7: Multiple superformula categories with intersection. Our approach correctly separates the three sub-manifolds, even though they all connect via a common seam.

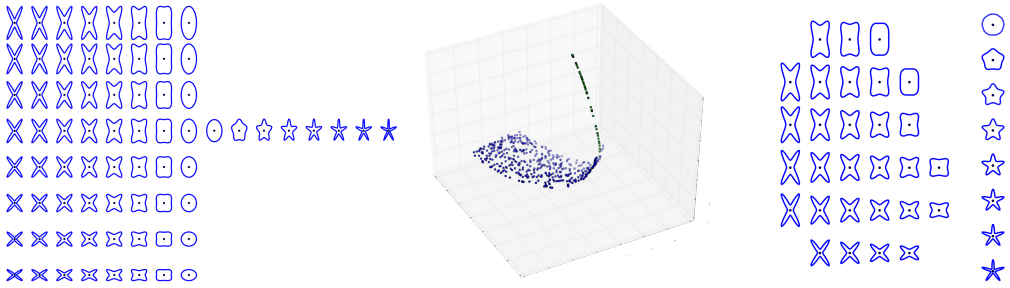
shown in Fig. 3.6.

3.6 Results and Discussion

We evaluated our method's performance at recovering various design space properties. We also compared how well different embedding approaches captured shape attributes.

3.6.1 Design Space Properties

We use the superformula examples to test the accuracy of our clustering algorithm and intrinsic dimension estimator. We conducted experiments with the number of clusters C set from 1 to 5, intrinsic dimension from 1 to 3, and three levels of linearity (curvature of the manifold). The dataset used in each experiment has a sample size



(a) Shapes in the superformula parameter space.

(b) Result of manifold clustering (as in Fig. 3.3, the design space $\mathcal{X}$ is visualized in three dimensions).

(c) Generated shapes in latent spaces.

Figure 3.8: Multiple superformula categories with intersection and different intrinsic dimensions. Our intrinsic dimension estimator automatically detects the appropriate dimensionality of the latent space for each design category (c).

of 500. All the experiments correctly separated the superformula shape categories and estimated the correct intrinsic dimensions (Fig. 3.7). In cases where multiple manifolds intersect (Fig. 3.7b), it is improper to use metrics like the rand index to evaluate clustering accuracy, because samples at the intersection can be classified to any adjacent group. For example, shapes turn into ellipses at the intersection (Fig. 3.7a), so it does not matter whether they belong to the four- or five-pointed star group. Figure 3.7c shows that our approach captures this case.

Figure 3.8 demonstrates a case where two manifolds with *different intrinsic dimensions* intersect: one superformula category with an intrinsic dimension of 1 intersects with another category with an intrinsic dimension of 2. Our method correctly separates the two categories and estimates the intrinsic dimension for each category, as shown in Fig. 3.8c.

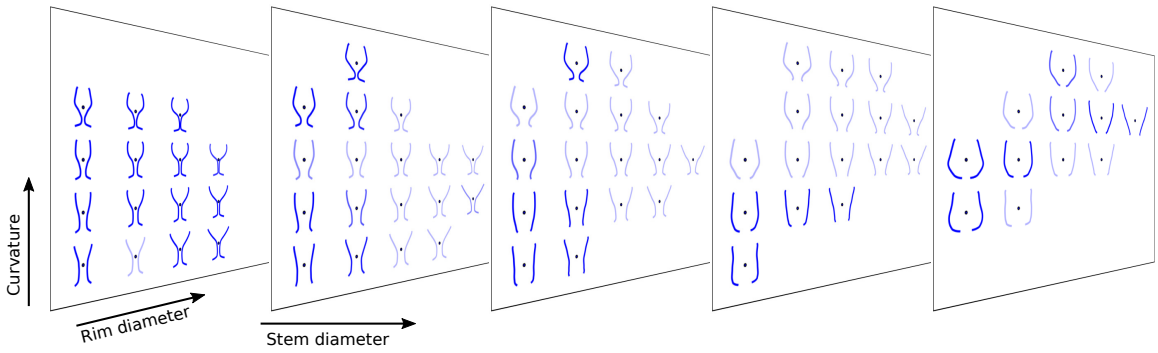


Figure 3.9: Synthesized glassware shapes in a 3D latent space. The embedding captured three shape attributes—the rim diameter, the stem diameter, and the curvature.

3.6.2 Principal Attributes

As mentioned above, a good embedding should precisely capture the right principal attributes. For superformula examples, we check this by comparing the shapes generated from the latent space $\mathcal{F}$ with those from the superformula parameter space $\mathcal{P}$. For the example shown in Fig. 3.7, the clustering separated shapes with a different number of arms, and each cluster's embedding correctly captured the two principal attributes—aspect ratio and roundness. In Fig. 3.8's example, as expected, the embedding captured the aspect ratio and the roundness attributes of the four-pointed stars, and just the roundness for the five-pointed stars.

Figure 3.9 shows the synthesized glassware in a 3-dimensional latent space. This embedding captured three attributes—the rim diameter, the stem diameter, and the curvature. With the variation of these three attributes, we synthesized a collection of shapes that generally covers all the training samples—wine, beer, champagne, and cocktail glasses. Similarly, the airfoil embedding shown in Fig. 3.10 also captured three attributes—the upper and lower surface protrusion, and the trailing edge direction.

For the airfoil example, the uncovered design manifold allows us to optimize airfoil

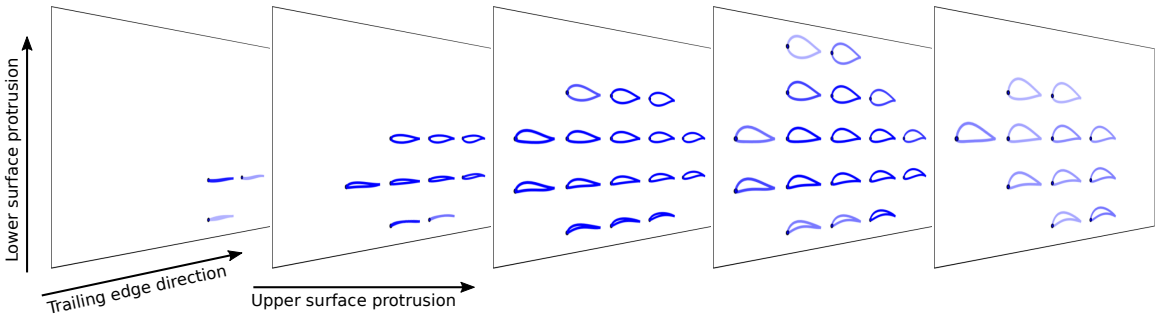
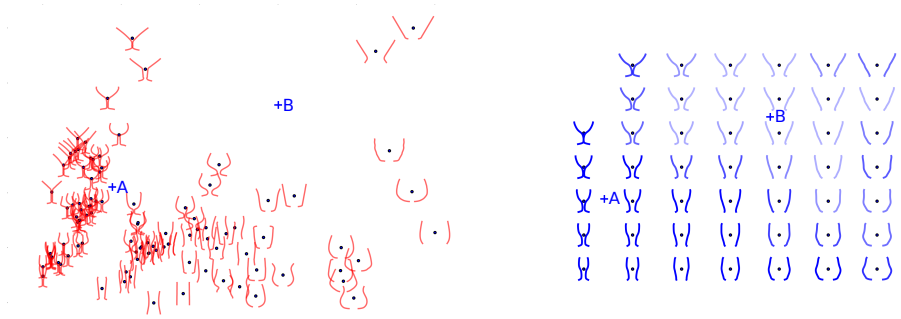


Figure 3.10: Synthesized airfoil shapes in a 3D latent space. The embedding captured three shape attributes—the upper and lower surface protrusion, and the trailing edge direction.



(a) Arrangement of training samples in the latent space (for simplicity, this is a 2D projection of the 3D latent space).

(b) Synthesized shapes.

Figure 3.11: Point A has high sample density, and thus higher confidence that synthesized shapes will look similar to nearby real-world samples. In contrast, point B has low sample density, and thus lower confidence but higher chance of generating an unusual or creative shape. Shade darkness correlates with higher local density.

shapes over this continuous low-dimensional space instead of the original geometric design space, using metamodeling techniques like Bayesian optimization. In our future work, we will compare the performance of metamodeling techniques with and without applying our dimensionality reduction method.

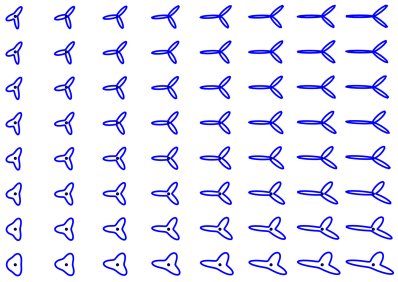
3.6.3 Are all regions on the manifold equally valid?

The different shades in Fig. 3.7–3.10 represent for local density of the shape collection (*i.e.*, training samples) distributed in the latent space $\mathcal{F}$. At positions nearby

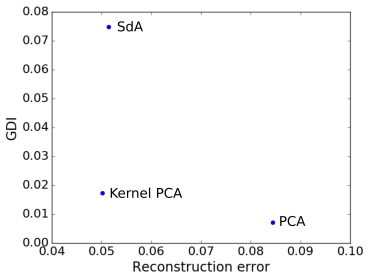
the training samples (*e.g.*, point A in Fig. 3.11), we plot the synthesized shape with a darker color. This means we are more certain of its validity because it lies closer to real-world training samples. In contrast, in locations where training samples are sparse (*e.g.*, point B in Fig. 3.11), we plot the synthesized shape with a lighter color. This means we are less certain of its validity, and our model may create shapes that diverge from the training samples. In the latent space where the training samples are absent—what we call a *design cavity*—new designs might be innovative or they may be unrealistic. For example, as shown in Fig. 3.9, inside regions where shapes have light colors, we synthesized glassware that is not similar to any of our training samples. In this case, the model generated innovative designs rather than unrealistic ones; however, to our knowledge there is no formal mathematical way of differentiating those two cases automatically. Doing so would be a compelling topic of future research.

3.6.4 Sample Arrangement in Latent Space

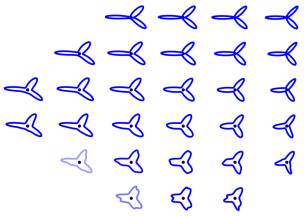
Figure 3.12 shows the comparison between results obtained from different embedding methods. For this superformula example, PCA has a high reconstruction error because the design space is nonlinear. This results in some abnormal shapes (*i.e.*, shapes with attributes that do not exist in the parameter space in Fig. 3.12a) in the latent space created by PCA (Fig. 3.12c). SdA and kernel PCA have similar reconstruction errors. However, the latent space created by kernel PCA (Fig. 3.12d) better aligns with the original parameter space than that of SdA (Fig. 3.12e), because SdA generates abnormal shapes (Fig. 3.12e, top right). Since these abnormal shapes have light colors, we know that they are inside the design cavity; while in fact if the embedding preserved the geodesic distances between samples (*i.e.*, low GDI), the



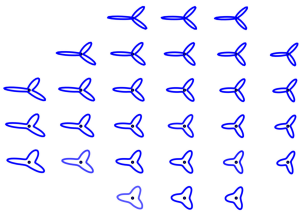
(a) Shapes in the superformula parameter space.



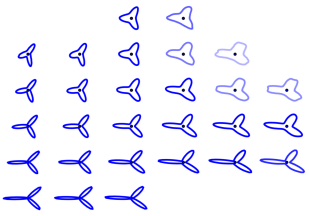
(b) Reconstruction error and geodesic distance inconsistency.



(c) Embedding and shape synthesis result by PCA.



(d) Embedding and shape synthesis result by kernel PCA.



(e) Embedding and shape synthesis result by a stacked denoising autoencoder.

Figure 3.12: Comparison of different embedding methods. The abnormal shapes generated by PCA and SdA are due to high reconstruction error and high GDI respectively.

sample arrangement should resemble that of PCA or kernel PCA, where there is no design cavity (Fig. 3.12c and 3.12d). Therefore both high reconstruction errors (*e.g.*, PCA in this example) and high GDI (*e.g.*, SdA in this example) can create abnormal or invalid shapes.

3.7 Summary

We introduced an approach to learn the inherent properties of a design space and evaluate two-way mappings between a design space and a latent space. By correctly identifying the design space properties such as the number of design categories and the intrinsic dimension, one can then create an embedding that precisely captures the principal attributes of each design category, assuming that the embedding is

well-chosen based on the reconstruction error and the pairwise distance preservation. This means the synthesized shapes will have no unexpected shape variation, missing diversity, or repeated shapes (brought about by unnecessary dimensions). We also introduced a benchmark for rigorously testing design embeddings that accounted for non-linear, multiple (potentially intersecting) manifolds of controllable intrinsic dimension. We encourage others to use this benchmark to improve future design embeddings.

While this chapter mainly addressed geometric design spaces, our approach would extend to any type of design space, including those that involve text, materials, or combinations with geometry (For an example of a combined material and geometry space). It could be applied to improving interfaces that help novices explore designs, aiding high-dimensional design optimization, and helping model consumer preferences in high-dimensional design spaces. Our work's main implication is that choosing a design embedding carries with it important choices about what you value in your latent space: Should it reconstruct designs consistently? Should it preserve the topology of the design space? What semantic attributes should the latent space capture? Choosing an embedding with the properties you want is not straightforward; our approach provides a principled way to compare and contrast embeddings—to help navigate those options and identify useful properties of both the embedding and your design space in general.

So far this dissertation has shown how to measure the intrinsic complexity of a design space, which guides data-driven design synthesis process. Purely data-driven design synthesis methods ignore designs' functional or geometrical constraints and characteristics, and thus will have limitations. The next two chapters will look at incorporating prior knowledge into data-driven models to improve the quality of design synthesis.

Chapter 4: Aerodynamic Design Synthesis, Optimization, and Shape Exploration

The work in this chapter was published in the AIAA SciTech 2019 Forum [42]

4.1 Introduction

Aerodynamic shape optimization is a necessary step in designing parts like aircraft wings and (propeller/rotor/turbine) blades. It has been an active research area for over 60 years [198]. The bottleneck of most global optimization methods for aerodynamic design is the computational cost of the computational fluid dynamics (CFD) simulations. To combat this, surrogate-based modeling approaches are used [4, 5, 76, 123, 146] to reduce the number of simulations by balancing exploration and exploitation while sampling the design space. However, the computational cost of sampling the design space increases exponentially with the dimensionality of the design space due to the curse of dimensionality [14]. Previous research has looked into dimensionality reduction (DR) of the original parametric design space (*i.e.*, the space of designs represented by shape parameters such as B-spline control points). This permits faster exploration by capturing only those dimensions that either affect the final design's performance [19–21, 88, 141] or capture major shape variability [55, 56, 223, 224, 239]. But these DR models may not accurately capture the true variation that we observed in real-world airfoils, *e.g.*, those in the UIUC

airfoils database. A vast amount of research on DR has been conducted in the field of machine learning, where deep neural networks such as variational autoencoders (VAEs) [117] and generative adversarial networks (GANs) [85] have successfully represented data from complex high-dimensional distributions, such as images, by using low-dimensional latent variables.

In this chapter, we address the problem of reducing the design space dimensionality for aerodynamic shape optimization. We apply GANs to learn an interpretable low-dimensional space (*i.e.*, the *latent space*) that encodes how aerodynamic shapes vary. To avoid the limitation caused by shape parameterizations (*e.g.*, curve-fitting errors and the lack of representation flexibility), we learn directly the distribution of points along the curves instead of curve parameters (such as Bézier control points). However, naïve application of neural network techniques to airfoil designs does not work well because the output is noisy and does not conserve important continuity properties important for aerodynamic shapes. Therefore, we use Bézier-GANs [46] to generate aerodynamic shapes. The Bézier-GAN model can be used for reducing the design parameters of any smooth shapes such as aerodynamic or hydrodynamic designs. The design optimization method demonstrated in this chapter is applicable to cases where the design space is reducible and represented by a lower-dimensional representation.

The specific scientific contributions of this paper are:

1. A new type of generative model appropriate for smooth geometry (such as those expressed via splines or Bezier curves) that improves the sample quality and convergence rate compared to traditional GANs.

2. A study of the comparative optima and convergence rate of several competing optimization methods—multiple parametric forms including our approach, Principal Component Analysis, PARSEC, and NURBS representations, along

with two optimization strategies (Bayesian Optimization and Genetic Algorithms)—and illuminate the performance conditions under which different approaches improve over others.

3. A two-stage optimization method that prioritizes the optimization of major shape attributes.

4.2 Background

In this section, we introduce previous work on common algorithms used in aerodynamic design optimization (Sec. 4.2.1), parameterization techniques (Sec. 4.2.2), and methods for reducing design space dimensionality (Sec. 4.2.3).

4.2.1 Optimization Methods

Aerodynamic design is, in large part, an optimization problem. One common objective is to find design variables that minimize the drag coefficient C_D, while maximizing or constraining the lift coefficient C_L [20, 88, 223]. There are primarily three approaches for solving the optimization problem: evolutionary algorithms (EA), surrogate-based optimization (SBO), and gradient-based methods.

Evolutionary algorithms (EA) are gradient-free optimization algorithms that mimic the process of biological evolution through mutation, recombination, and reproduction of different designs. Genetic algorithms (GA), a type of EA, is widely used in aerodynamic shape optimization [56, 106, 224]. Work has also been done to augment GA with the Bees algorithm [211] and adaptive mutation rates [104], resulting in more accurate optimization and/or faster convergence. Other EA methods applied in aerodynamic optimization are differential evolution [139] and particle swarm optimization [173, 219]. However, due to the large number of function calls needed in

each generation, EAs can be prohibitively expensive computationally, especially if every evaluation requires a high-fidelity computational fluid dynamics simulation.

Surrogate-based optimization (SBO) uses an inexpensive surrogate model to approximate the expensive function of the quantity of interest (QoI) (*i.e.*, the optimization objective). Bayesian optimization (BO) is a commonly used method for SBO. It consists of two components — a sampling method (*e.g.*, maximum expected improvement [108] or maximum upper confidence bound [205]) and a surrogate modeling method (*e.g.*, Gaussian process regression, also know as kriging [171]). In each iteration, the sampling method samples a point in the design space for evaluation of the QoI, and then that point and its QoI update the surrogate model. Compared to methods like genetic algorithms, surrogate-based optimization reduces the number of expensive CFD evaluations needed in aerodynamic shape optimization [20, 55, 92, 141, 224]. However, for a high-dimensional design space, the number of evaluations will still be inevitably high due to the curse of dimensionality [14, 174]. Note that in these cases, kriging can also be prohibitively expensive at the later stage when the model is trained on a large number of evaluated samples since its computational cost scales cubically with the sample size (though practical approximation methods do exist to reduce this cost).

Gradient-based methods search for the optimal solution based on the gradient of the objective function. When the objective is based on CFD simulations, automatic differentiation (AD)—a generalization of adjoint methods used by the CFD community—is usually used to compute the gradients. It provides a relatively fast and exact method of calculating numerical gradients. The computer records every elemental operation used to calculate a QoI ("forward pass") before reversing through this "tape" to determine the sensitivity of the QoI with respect to each parameter. Generally, gradient calculations are exact and have a computational

cost within an order of magnitude of the forward pass. Because of this, previous work [65, 112, 145, 184, 185, 213, 225] have used AD for gradient calculations. Combined with optimization algorithms such as SQP [65, 225], steepest descent [184], and Newton- and quasi-Newton methods [112, 145, 185], AD can drastically accelerate gradient calculations in the optimization process, even in complex or turbulent models [145, 185, 225].

However, for optimization using state-of-the-art turbulence models such as Large Eddy Simulation, one cannot use adjoint methods because the chaotic butterfly exponential divergence of trajectories makes the adjoint ill-posed [129]. In addition, an AD gradient is only applicable at one point; thus, unlike *e.g.*analytical derivatives, where a single equation provides exact derivatives at any point, AD requires a forward pass before each new gradient calculation, contributing to a large portion of the optimization cost. In terms of memory, building the tape of operators can be expensive compared to, *e.g.*, a finite difference method. Additionally, as a method of gradient calculation, AD will still maintain the disadvantage inherent in gradient-based algorithms, *e.g.*, converging to local minima. As a workaround solution, Berguin *et al.* [21] use solutions to SBO as starting points for AD methods, hoping to find good local optima.

4.2.2 Shape Parameterization

Parameterization maps a set of parameters to points along a smooth curve or surface via a parametric function. Common parameterization for aerodynamic shapes includes splines (*e.g.*, B-spline and Bézier curves) [105, 178, 218], free-form deformation (FFD) [113, 186], class-shape transformations (CST) [126, 152], PARSEC [138, 202], and Bézier-PARSEC [63]. While this work does not study paramterization, we show

the optimization performance of two paramterization approaches, namely nonuniform rational B-splines (NURBS) [134] and PARSEC [138], in comparison to our proposed method.

Usually during design optimization, parameters are sampled to generate design candidates [92]. There are two main issues when optimizing these parameters from conventional parameterization: (1) one has to guess the limits of the parameters to form a bounding-box within which the optimization operates, and (2) the design space dimensionality is usually higher than the underlying dimensionality for representing sufficient shape variability [49] —*i.e.*, to capture sufficient shape variation, manually designed shape parameterizations require higher dimensions than are strictly necessary.

4.2.3 Design Space Dimensionality Reduction

It is computationally expensive to search for solutions in the design space directly due to the space's high dimensionality. Factor screening methods [151, 230] are used to select the most relevant design variables for a design problem while fixing the rest as constant during optimization. These methods fail to consider the correlation between design variables. Thus, researchers have found ways to capture the low-dimensional subspace that identifies important directions with respect to the change of *response* (*i.e.*, QoI or performance measure) [19–21, 88, 141]. This response-based dimensionality reduction usually has several issues: 1) it requires many simulations when collecting samples of response gradients; 2) variation in gradients can only capture non-linearity rather than variability in the response, so extra heuristics are required to select latent dimensions that capture steep linear response changes; 3) the learned latent space is not reusable for any different design space exploration or

optimization task (*i.e.*, when a different response is used); and 4) the linear DR techniques applied in previous work may fail on responses with a non-linear correlation between partial derivatives.

The first three issues can be avoided by directly applying DR on design variables without associating them with the response. Note that by doing this, we are assuming that if changes in a design are negligible, changes of responses are also negligible. In the area of aerodynamic design, researchers use linear models such as proper orthogonal decomposition (POD), also known as principal component analysis (PCA) [55, 56, 239], and nonlinear models like generative topographic mapping [223, 224] to reduce the dimension of design variables. More work on DR has been done in other fields such as image processing and computer graphics [84, 132], where DR is used for generating and visualizing data. Deep neural networks such as VAEs and GANs have been widely applied in these areas to learn the latent representation of data. These methods are known for their ability to learn complex high-dimensional data distributions. Our work extends this class of techniques by considering the generation of smooth geometries such as those needed in spline-based representations.

Note that as DR models map latent variables to shapes, we can treat the latent variables and the mapping as parameters and the parametric function. Thus, in a broader sense, we will also refer to these methods as parameterization in Sec. 4.6.

4.3 Obtaining Disentangled Latent Representation Using Generative Adversarial Networks

We use a method based on GANs [85] to train a generative model that synthesizes aerodynamic shapes from interpretable low-dimensional latent codes. GANs are one

type of deep neural network architecture which consists of two components: a generator and a discriminator. The generator takes in random noise from some known prior distribution $P_{\boldsymbol{z}}$. Its objective is to generate samples from the desired distribution (*i.e.*, data distribution P_{data}). The discriminator takes in a sample (either from the training data or synthesized by the generator) and predicts the probability of the sample coming from the training data. The generator tries to make the generative distribution P_G look like P_{data} to fool the discriminator; the discriminator tries not to be fooled. GANs achieve this by minimizing the objective:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))] \tag{4.1}$$

where D is the discriminator, and G is the generator. Both components improve via training until the discriminator cannot differentiate between real and fake inputs, implying that the generative distribution resembles the data distribution. A trained generator thus can map from the predefined noise distribution to the distribution of designs. The noise input $\boldsymbol{z}$ is considered as the latent representation in the dimensionality reduction scenario, since $\boldsymbol{z}$ captures the variability of the data.

Standard GANs are not built for learning latent representations; thus, they cannot be used to reduce the dimensionality of the design space. To compensate for this weakness, InfoGANs [52] encourage interpretable latent representations by maximizing the mutual information between some noise variables (called *latent codes*) and the generated samples. Thus, InfoGAN's generator takes both latent codes $\boldsymbol{c}$ and random noise $\boldsymbol{z}$ as inputs. Unfortunately, it is hard to directly maximize the mutual information $I(\boldsymbol{c}; G(\boldsymbol{c}, \boldsymbol{z}))$, so instead an InfoGAN approximates the solution by maximizing a lower bound. In practice, this is realized by adding an extra fully connected layer to the discriminator to predict the latent codes. Please refer to Ref. [52] and Sect. 5.2.3

for more details about the InfoGAN. We build upon this line of work below, extending it to spline-based geometry.

4.4 Spline-Based Shape Synthesis

Typical approaches to generative shape models (such as GANs) represent shapes as a collection of discrete samples (*e.g.*, as pixels or voxels) owing to their original development in the computer vision community. For example, a naïve way of synthesizing shapes like airfoils would be to generate this *discrete representation* directly using the generator, such as generating a fixed number of coordinates sampled along the airfoils boundary curve (*e.g.*, Fig. 4.2, right). However, in practice, airfoils typically possess substantial smoothness/continuity and are typically represented using parametric curve families like splines, Bézier curves, or NURBS surfaces. The naïve GAN representation of predicting discretized curves from the generator usually (1) creates noisy curves that have low smoothness and (2) has parametric outputs that are harder for humans to interpret and use in standard CAD packages compared to equivalent curve representations (*e.g.*, Bézier curves). This creates problems, particularly in aerodynamic shape synthesis.

To solve this issue, we modified the InfoGAN's generator such that it only generates smooth shapes that conform to Bézier curves. We call this generative adversarial network a Bézier-GAN [46]. As shown in Fig. 4.1, most of its architecture is adapted from the InfoGAN. However, before outputting discrete coordinates along the curve, the generator synthesizes *control points* P, *weights* w, and *parameter variables* t of rational Bézier curves. The last layer—the Bézier layer—converts this rational Bézier

curve representation into discrete representation X:

$$X_j = \frac{\sum_{i=0}^{n} \binom{n}{i} t_j^i (1-t_j)^{n-i} P_i w_i}{\sum_{i=0}^{n} \binom{n}{i} t_j^i (1-t_j)^{n-i} w_i}, \qquad j = 0, ..., m \tag{4.2}$$

where n is the Bézier degree, and the number of discrete points to represent the curve is $m+1$. Since variables $\{P_i\}$, $\{w_i\}$, and $\{t_j\}$ are differentiable in Eq. 4.2, we can train the network using back propagation. Figure 4.2 compares synthesized shapes with and without using a Bézier layer.

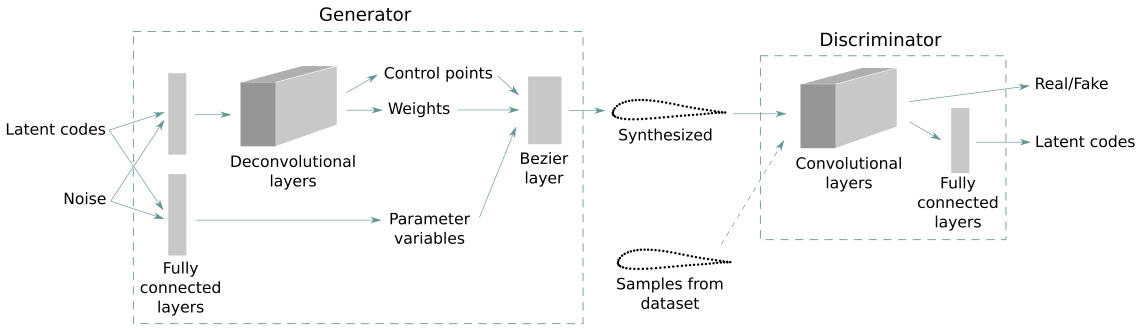


Figure 4.1: Model architecture of the Bézier-GAN.

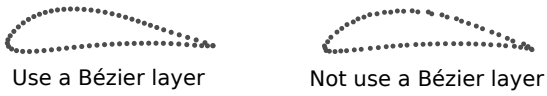


Figure 4.2: Synthesized airfoils using a generator with and without a Bézier layer.

4.5 Optimization over the Learned Latent Space

4.5.1 The Optimization Problem in the Latent Space

The optimal aerodynamic shape can be solved by $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$, where $\boldsymbol{x}$ is an aerodynamic shape (expressed in this case by the latent codes and the Bézier curve parameters) and $f(\boldsymbol{x})$ is some performance measure defined over $\boldsymbol{x}$ (*e.g.*, lift,

drag, *etc.*). Since the function f is usually non-convex, methods such as EA or SBO are often used for optimization [62, 70, 77, 249]. These methods search for the global optimum by exploring the design space $\mathcal{X}$. However, since $\mathcal{X}$ is usually high-dimensional, it takes many performance evaluations to find the optimal solution due to the curse of dimensionality [14]. Since we can modify $\boldsymbol{x}$ by changing the latent code $\boldsymbol{c}$, which has a lower dimension than $\boldsymbol{x}$, finding the optimal shape $\boldsymbol{x}^*$ is equivalent to finding an optimal latent code $\boldsymbol{c}^*$. Thus, we solve the following problem instead:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} h(\boldsymbol{c}) = \arg\min_{\boldsymbol{c}} f(\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}[G(\boldsymbol{c}, \boldsymbol{z})]) \tag{4.3}$$

and then use $\boldsymbol{c}^*$ to synthesize the optimal shape $\boldsymbol{x}^* = \mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}[G(\boldsymbol{c}^*, \boldsymbol{z})]$.

We can then apply SBO to find $\boldsymbol{c}^*$ instead of $\boldsymbol{x}^*$. There are two major components in SBO: an inference model and an acquisition function. In this chapter we use a SBO method called Efficient Global Optimization (EGO) [108], which uses Gaussian process (GP) regression [171] as the inference model, and expected improvement (EI) [108] as the acquisition function:

$$\mathbb{E}[I(\boldsymbol{c})] = \mathbb{E}[\max(h_{min} - h(\boldsymbol{c}), 0)] \tag{4.4}$$

where h_{min} is the current best function value. At each step t of EGO we want to find $\boldsymbol{c}^{(t)}$ that is expected to best improve upon the current optimal solution:

$$\boldsymbol{c}^{(t)} = \arg\max_{\boldsymbol{c}} \mathbb{E}[I(\boldsymbol{c})] \tag{4.5}$$

Now with the ingredients of GP regression and EI, the EGO process simply repeats the following steps:

1. Estimate the function h by using GP regression;

2. Compute EI using Eq. 4.4 and the learned GP model;

3. Search for the point $\boldsymbol{c}^{(t)}$ that has the highest EI (solving Eq. 4.5);

4. Evaluate the function h at $\boldsymbol{c}^{(t)}$, and add the new $(\boldsymbol{c}^{(t)}, h(\boldsymbol{c}^{(t)}))$ pair into the dataset for learning GP regression.

4.5.2 Unbounded Bayesian Optimization

To find $\boldsymbol{c}^{(t)}$ in Step 3, we can use gradient-based optimization algorithms like BFGS [25, 74, 81, 192]. However, these methods may get stuck in local optima and are unstable if operated in an unbounded space (*i.e.*, the solution may go too far from feasible regions and thus will diverge). Random search is a simple alternative approach that searches $\boldsymbol{c}^{(t)}$ within fixed variable bounds; however, the optimal solution may be located outside those bounds. To circumvent these issues, we search for the solution near $\boldsymbol{c}^{(t-1)}$ (*i.e.*, the point evaluated at step $t-1$) without requiring a boundary. Specifically, we search for $\boldsymbol{c}^{(t)}$ among samples drawn from the distribution $\mathcal{N}(\boldsymbol{c}^{(t-1)}, \sigma^2)$, where σ controls the dispersion of the drawn samples (Fig. 4.3). Combined with the EI criteria, each iteration of the search area moves in the direction which is expected to improve the current optimal solution. This moving search area eliminates the limitation of variable bounds. A larger σ encourages exploration and prevents the solution from getting stuck in local optima while a smaller σ encourages exploitation and refines the current optimal solution. We use a decreasing σ over iterations (*i.e.*, in each iteration, multiply σ by a constant γ that is close to but smaller than 1), so that the algorithm first explores then focuses more on exploitation.

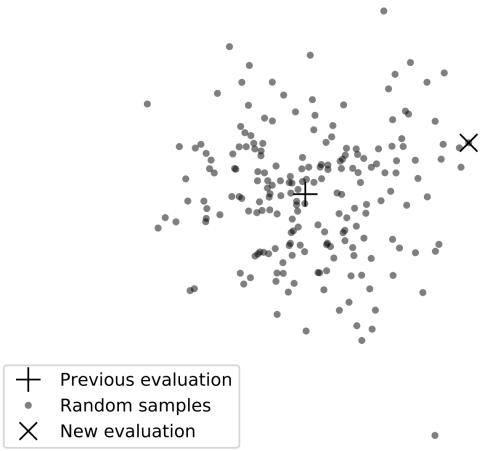


Figure 4.3: Unbounded sampling in Bayesian optimization.

4.6 Experiment: Airfoil Synthesis and Shape Optimization

In this section, we demonstrate our method via an airfoil optimization task. Rather than targeting one specific airfoil model (*e.g.*, the NACA 0012 airfoil in Ref. [88] or the RAE 2822 airfoil in Ref. [224]) and its perturbations, we search for the optimal design within all the existing major airfoil models. We show that Bézier-GAN learns realistic shape variations from these airfoil models and that optimizing in the latent space accelerates convergence.

4.6.1 Dataset and Preprocessing

We use the UIUC airfoil database[1] as our training data for the Bézier-GAN. It provides the geometries of nearly 1,600 real-world airfoil designs, each of which is represented by discrete coordinates along their upper and lower surfaces. The number

[1] http://m-selig.ae.illinois.edu/ads/coord_database.html

of coordinates for each airfoil is inconsistent across the database, so we use B-spline interpolation to obtain consistent shape representations. Specifically, we interpolate 192 points over each airfoil with the concentration of these points along the B-spline curve based on the curvature [105]. The preprocessed data are visualized at the top of Fig. 4.4.

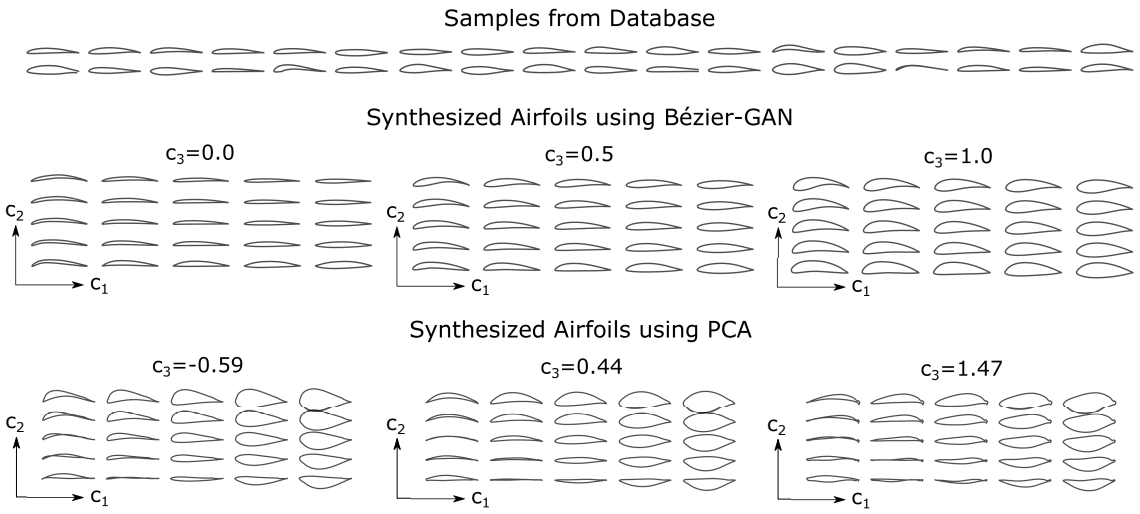


Figure 4.4: Examples in the airfoil database and synthesized airfoil shapes in three-dimensional latent spaces (visualized by uniform slices of multiple two-dimensional spaces).

4.6.2 Dimensionality Reduction

We build a Bézier-GAN model based on the architecture in Fig. 4.1. The latent codes are from a three-dimensional uniform distribution, and the input noise is from a ten-dimensional Gaussian distribution. In the discriminator, we use six one-dimensional convolutional layers followed by fully connected layers to predict latent codes and the probability of the input data coming from the dataset. For the generator, we use three one-dimensional deconvolutional layers [243] to predict the control points $\{P_i|i = 0, ..., n\}$ and the weights $\{w_i|i = 0, ..., n\}$, and three fully connected layers followed by a softmax activation to predict discrete differences between

parameter variables $\{t_{j+1} - t_j | j = 0, ..., m-1\}$. Interested readers can refer to detailed network architectures and hyperparameters in our Tensorflow implementation available on Github[2].

We optimize the Bézier-GAN using an Adam optimizer [116] and train it on an Nvidia Titan X GPU. The wall-clock training time is about 1 hour, and the inference takes less than 15 seconds.

Figure 4.4 shows synthesized airfoil shapes by linearly interpolating points in the latent space. The middle subplot shows that airfoils synthesized by Bézier-GAN are realistic and capture most variation in the airfoil dataset. We also obtained an interpretable latent space: the horizontal axis (c_1) captured the leading edge angle, the vertical axis (c_2) captured the trailing edge angle, and the third axis (c_3) captured the thickness.

We use PCA as a baseline DR method to compare the synthesis quality. The latent space is also set to three-dimensional. The results of PCA are shown at the bottom of Fig. 4.4[3]. Compared to Bézier-GAN, PCA shows the limitations of a linear DR model by synthesizing unrealistic designs in some regions of the latent space.

4.6.3 Optimization

Our optimization objective is to maximize the lift to drag ratio C_L/C_D. We use XFOIL [67] to compute the lift and drag coefficients C_L and C_D.[4] The XFOIL operation conditions are set as follows: Reynolds number $Re = 1.8 \times 10^6$, Mach number $Ma = 0.01$, and angle of attack $\alpha = 0°$.

[2] https://github.com/IDEALLab/airfoil-opt-gan

[3] The bounds of the visualized latent space are based on the latent coordinates of the data, *i.e.*, the minimum bounding box for data points projected onto the latent space.

[4] We use XFOIL here to demonstrate our scientific contributions, however, our approach is not limited to XFOIL. One can apply our techniques to any CFD or performance code including RANS or LES.

For Bézier-GAN and PCA, we apply EGO on the three-dimensional latent spaces and use the trained Bézier-GAN generator or the inverse transformation of PCA to synthesize the optimal airfoils corresponding to the optimal latent codes.

We also compare these results to optimizing directly in the parametric design space. Specifically, we use two parameterizations, NURBS and PARSEC, and two optimization algorithms, EGO and GA, as additional experiments. We use the NACA 0012 airfoil as the initial design. The NURBS parameterization is based on Ref. [224]. The design space is defined as a ± 0.1 perturbation of the initial NURBS control point coordinates or a 20% perturbation of the initial PARSEC parameters. The population size of the GA is 100, and the chance of mutation (*i.e.*, the probability of mutating an individual's parameter) is 0.1. In each generation, we choose 30 best and 10 random individuals for crossover, and produce 5 children for each pair. We direct interested readers to our code for more details.

We run each experiment so that the total number of C_L/C_D evaluations is 1000. The results of each experiment setting are averaged over 10 runs. Figure 4.5 shows the best-so-far C_L/C_D versus the number of evaluations. It shows that the value reached in 100 XFOIL evaluations by Bézier-GAN+EGO takes other methods at least 500 XFOIL evaluations to reach. Figure 4.6 shows the optimal airfoils for all experiment settings. Runs from the same scenario are plotted on the same subplot. For PCA+EGO and NURBS+GA, the final optimal solutions are inconsistent compared to other methods, indicating the optimization converged to different local optima. The values of maximal C_L/C_D after 100 and 1000 evaluations are shown in Table 4.1.

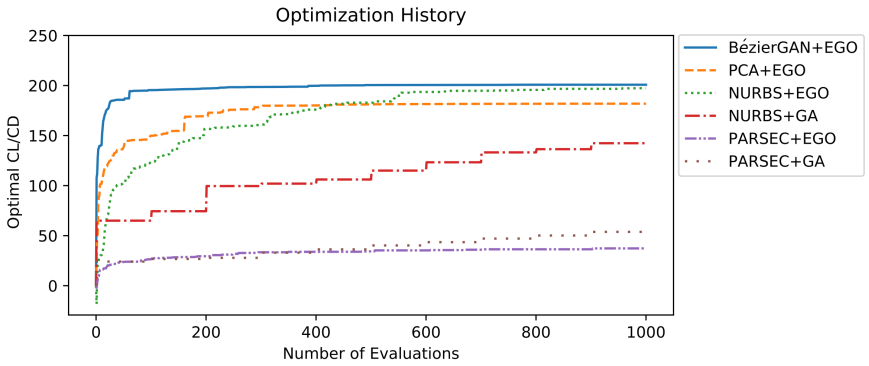


Figure 4.5: Optimization history (averaged over 10 runs).

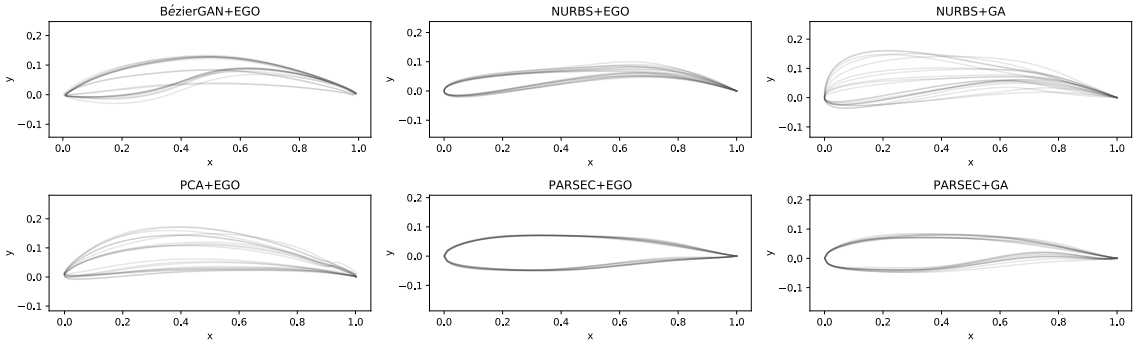


Figure 4.6: Optimal airfoils (airfoils in the same subplot are under the same experimental configuration but different runs).

4.6.4 GA Refining

Figure 4.5 shows that when using Bézier-GAN, the optimal C_L/C_D stops improving after 100 evaluations, whereas the optimal C_L/C_D improves continuously, though slowly, when using the NURBS parameterization. This is because the three-dimensional latent space does not contain as much shape variation as the NURBS design space. However, while the three-dimensional latent space captures major shape variations, minor shape variations are captured by the noise space (*i.e.*, the space of the random input noise of Bézier-GAN). Therefore, we can further search for an improvement in that noise space. We achieve this by using the optimal solution of EGO

Table 4.1: Values of C_L/C_D for optimal solutions.

# Eval.	BézierGAN+EGO	PCA+EGO	NURBS+EGO	NURBS+GA
100	**195.41 ± 2.94**	149.86 ± 13.46	122.75 ± 20.41	64.97 ± 4.82
1000	**200.80 ± 2.12**	181.86 ± 21.87	197.37 ± 3.22	142.35 ± 20.38

# Eval.	PARSEC+EGO	PARSEC+GA
100	26.44 ± 3.88	24.11 ± 0.90
1000	27.26 ± 4.07	53.77 ± 3.24

after 100 evaluations as the initial design and run GAs in both the latent space and the noise space. We call this *GA refining*. Specifically, we allow larger shape variation on the noise variables while limiting the variation on the latent variables during mutation. The results are shown in Figs. 4.7 and 4.8. In this way, the optimal C_L/C_D keeps improving even after the latent space is exploited.

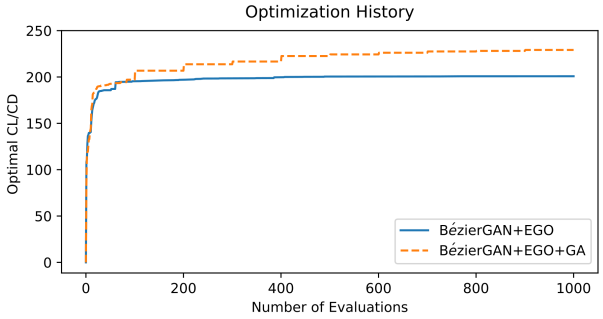


Figure 4.7: Optimization history for BézierGAN+EGO with and without GA refining.

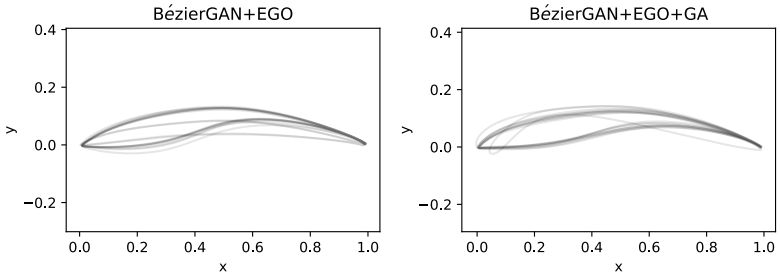


Figure 4.8: Optimal airfoils for BézierGAN+EGO with and without GA refining.

4.7 Summary

We use a Bézier-GAN to capture a low-dimensional latent space that encodes major shape variability of aerodynamic designs. Design optimization can then be conducted in this latent space to reduce the number of evaluations required to find the optimal solution. Our results show that our Bézier-GAN method significantly accelerates convergence and finds optimal designs that are comparable to those found by other algorithms.

Because the latent space discards minor variability in designs that can potentially contribute to higher performance, the final optimal solution may be not as good as directly optimizing in the design space given a sufficient number of evaluations. The GA refining mitigates this issue by continuing to explore the input noise space of the GAN after discovering good latent variables. There are other ways to improve the optimal solution while maintaining fast convergence. For example, the optimal solution obtained by our method can be used as a good start point for gradient-based optimization methods (*e.g.*, as in Berguin *et al.* [21]). For future research, we can concatenate a trained Bézier-GAN generator and an automatic differentiation solver to obtain the gradient of a QoI with respect to each latent variable directly. The low-dimensional gradients can then be applied to solve optimization problems.

Different from previous DR research for aerodynamic shape optimization which only targets one specific QoI (*i.e.*, response-based DR) or one airfoil model, the learned latent space in this work is reusable for optimizing any QoI for any airfoil model included in the UIUC airfoil database.

So far this dissertation has shown how to measure the intrinsic complexity of a design space to guide data-driven design synthesis and how to constrain the data-driven model to generate valid aerodynamic shapes. The next chapter will introduce

a design synthesis model that considers the inner-part dependencies of designs, which is another example of incorporating prior knowledge into data-driven design synthesis models.

Chapter 5: Synthesizing Designs with Inter-Part Dependencies

Portions of the work in this chapter were accepted to the Journal of Mechanical Design [48] and published in the ASME International Design Technical Conference [50]

5.1 Introduction

Representing a high-dimensional design space with a lower-dimensional *latent space* makes it easier to explore, visualize, or optimize complex designs. This often means finding a *latent representation*, or a *manifold*, along which valid designs, such as geometries, lie [44, 49].

While this works well for single parts, designs usually have multiple parts with inter-part dependencies. For example, the size and position of a conduit, lightening, or alignment hole in an airframe structure depend on the shape of the airfoil. Here we assume that design components are synthesized sequentially and do not consider bi-directional inter-part dependencies, although this kind of dependency exists in some design tasks (*e.g.*, alternating optimization of Part A and Part B in a design). We leave this topic for future research. Thus we can describe inter-part dependencies in a design by using a *directed acyclic graph* (DAG). This DAG captures whether the geometry of a part depends on the geometry of its parent part(s). In this case, one may want to identify first the *parent manifold* that captures the major variation of parent shapes, and then the *child manifold* that captures the major variation of feasible child

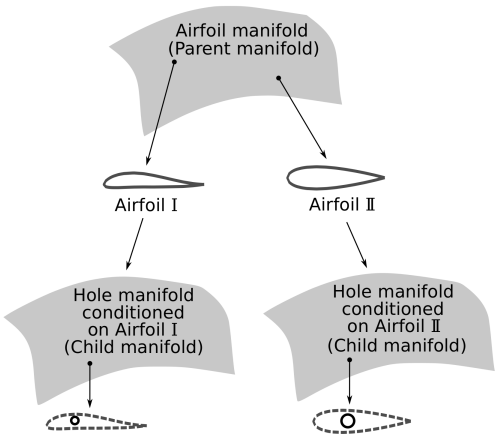


Figure 5.1: Manifolds of parent and child shapes

shapes conditioned on any parent shape (Fig. 5.1). Because, for example, we may first optimize the airfoil shape on the airfoil manifold (parent) to obtain the optimal lift and drag; and then given the optimal airfoil, we may optimize the hole's size and position on the hole manifold (child) for other considerations like light-weighting, *etc.*

This chapter addresses the problem of capturing the latent representation for multi-components designs. However, finding individual part manifolds that both represent the design space well, while also satisfying part configuration, is non-trivial. Traditionally, to learn the inter-part dependency, one has to either define explicit constraints [120, 121] or learn implicit constraints via adaptive sampling [44, 45]. The former uses hard-coded (often application-specific) constraints and hence lacks flexibility; whereas the latter queries external sources by human annotation, experiment, or simulation, and thus is expensive. In this chapter, we solve these problems by instead learning these constraints given examples. We assume that we only have the prior knowledge on inter-part dependencies, but not the specific types of constraints (*e.g.*, concentric, alignment, tangent, *etc.*) that confine the geometry of each part. We do this by identifying different levels of manifolds, where the higher-level (parent) manifold imposes implicit constraints on the lower-level (child) manifolds.

We propose a deep generative model that synthesizes designs in a hierarchical manner according to those inter-part dependencies: it first synthesizes parent parts, and then synthesizes parts conditioned on those parent parts and so on. At each level, the model simultaneously captures a parent manifold and an infinite number of child manifolds which are conditioned on parent parts. This results in latent spaces that can synthesize and search each part individually as well as their assemblies. Importantly, our method is fully data-driven and requires no hard-coded rules or querying external sources, except for providing an example dataset of designs with part correspondence and known inter-part dependencies. For designs without part correspondence, we can apply unsupervised co-segmentation [194] as a pre-processing step. This work facilitates the understanding of complex design spaces where inter-part dependencies exist and can be used for efficient hierarchical design space exploration.

The chapter's key contributions are as follows:

1. A novel deep generative model architecture that simultaneously learns a design's inter-part dependencies and each part's geometry variation conditioned on the corresponding parent part(s). It decouples each part's latent space so that we can perform design space exploration in separate low-dimensional latent spaces.

2. New benchmark datasets—both real-world and synthetic—that can be used for studying different kinds of inter-part dependencies including: type of geometric constraints, depth of hierarchy, and branching factor of parent/child relationships. This dataset can aid in the future evaluation of generative models of hierarchical parts.

3. Characterizing the effects of sample size and part dependencies' complexity (depth and branching factor) on the synthesis performance of our generative model.

4. A new evaluation metric for generative models that measures the consistency of shape variation in the latent space.

5.2 Related Work

Our work produces generative models that synthesize designs from latent representations. There are primarily two streams of related research from the fields of engineering design and computer graphics—Specifically, design space dimensionality reduction and design synthesis. We also review generative adversarial networks (GANs) [85], which we use to build our model.

5.2.1 Design Space Dimensionality Reduction

While designs can be parametrized by various techniques [179], the number of design variables (*i.e.*, the dimensionality of a design space) increases with the geometric variability of designs. In tasks like design optimization, to find better designs we usually need a design space with higher variability, *i.e.*, higher dimensionality. This demand creates the problem of exploring a high-dimensional design space. Based on the curse of dimensionality [14], the cost of exploring the design space grows exponentially with its dimensionality. Thus, researchers have studied approaches for reducing the design space dimensionality. Normally, dimensionality reduction methods identify a lower-dimensional latent space that captures most of the design space's variability. This can be grouped into linear and non-linear methods.

Linear dimensionality reduction methods select a set of optimal directions or basis functions where the variance of shape geometry or certain simulation output is maximized. Such methods include the Karhunen-Loève expansion (KLE) [51, 66], principal component analysis (PCA) [69], and the active subspaces approach [214].

In practice, it is more reasonable to assume that design variables lie on a non-linear manifold, rather than a hyper-plane. Thus, researchers also apply non-linear methods to reduce the dimensionality of design spaces. This non-linearity can be achieved by (1) applying linear reduction techniques locally to construct a non-linear global manifold [69, 130, 167–169]; (2) using kernel methods with linear reduction techniques (*i.e.*, using linear methods in a Reproducing Kernel Hilbert Space that then induces non-linearity in the original design space) [49, 69]; (3) latent variable models like Gaussian process latent variable model (GPLVM) and generative topographic mapping (GTM) [224]; and 4) neural networks based approaches such as self-organizing maps [165] and autoencoders [27, 49, 59, 69].

This work differs from these past approaches in that we aim at identifying two-level latent spaces with the lower-level encodes inter-part dependencies, rather than learning only one latent space for the complete design.

5.2.2 Data-Driven Design Synthesis

Design synthesis methods can be divided into two categories: rule-based and data-driven design synthesis. The former (*e.g.*, grammars-based design synthesis [80, 120, 121]) requires labeling of the reference points or surfaces and defining rule sets, so that new designs are synthesized according to this hard-coded prior knowledge; while the latter learns rules/constraints from a database and generates plausible new designs with similar structure/function to exemplars in the database.

Usually, dimensionality reduction techniques allow inverse transformations from the latent space back to the design space, thus can synthesize new designs from latent variables [49, 51, 59, 69]. For example, under the PCA model, the latent variables define a linear combination of principal components to synthesize a new

design [51]; for local manifold based approaches, a new design can be synthesized via interpolation between neighboring points on the local manifold [130]; and under the autoencoder model, the trained decoder maps any given point in the latent space to a new design [27, 59]. Researchers have also employed generative models such as kernel density estimation [209], Boltzmann machines [99], variational autoencoders (VAEs) [153], and generative adversarial nets (GANs) [137, 231] to learn the distribution of samples in the design space, and synthesize new designs by drawing samples from the learned distribution. Discriminative models like deep residual networks [197] are also used to generate 3D shapes.

These aforementioned models synthesize a design or a shape as a whole. There are methods that synthesize new shapes by assembling or reorganizing parts from an existing shape database, while preserving the desired structures [36, 109, 208, 234, 246]. The shapes are usually parametrized by high-level abstract representations, such as hand-crafted feature vectors [109] or shape grammars [208]. While these methods edit shapes at a high-level, they do not control the local geometry of each synthesized component.

Previously the inter-part dependencies of shapes have been modeled by grammar induction [208], kernel density estimation [73], probabilistic graph models [36, 99, 109], and recursive autoencoders [137]. Those methods handle part relations and design synthesis separately. In contrast, our method encodes part relations through the model architecture, so that it simultaneously learns the inter-part dependencies and single part geometry variation. The model can also be used for inferring the generative distribution of each part conditioned on any parent part.

5.2.3 Generative Adversarial Networks

As introduced in Sect. 4.3, generative adversarial nets [85] model a game between a generative model (*generator*) and a discriminative model (*discriminator*). One advantage GANs hold over Variational Auto-encoders (VAEs) [117] is that GANs tend to generate more realistic data [166]. But a disadvantage of the original GAN formulation is that it cannot learn an interpretable latent representation. Built upon these "vanilla" GANs, the InfoGAN [52] aims at regularizing the latent representation of the data space by maximizing a lower bound of the mutual information between a set of *latent codes* $\boldsymbol{c}$ and the generated data. The generator is provided with both $\boldsymbol{z}$ and $\boldsymbol{c}$. Thus, the generator distribution $P_G = G(\boldsymbol{c}, \boldsymbol{z})$ is conditioned on $\boldsymbol{c}$. The mutual information lower bound L_I is

$$L_I(G, Q) = \mathbb{E}_{\boldsymbol{c}\sim P(\boldsymbol{c}), \boldsymbol{x}\sim G(\boldsymbol{c},\boldsymbol{z})}[\log Q(\boldsymbol{c}|\boldsymbol{x})] + H(\boldsymbol{c}) \tag{5.1}$$

where $H(\boldsymbol{c})$ is the entropy of the latent codes, and $Q(\boldsymbol{c}|\boldsymbol{x})$ is called the auxiliary distribution which approximates $P(\boldsymbol{c}|\boldsymbol{x})$. We direct interested readers to [52] for the derivation of L_I. The InfoGAN objective combines L_I with the standard GAN objective:

$$\min_{G,Q} \max_{D} V_{InfoGAN}(D, G, Q) = V(D, G) - \lambda L_I(G, Q) \tag{5.2}$$

where λ is a weight parameter.

In practice, $H(\boldsymbol{c})$ is treated as constant if the distribution of $\boldsymbol{c}$ is fixed. The auxiliary distribution Q is parametrized by a neural network—here we call it the *auxiliary network*.

In our design synthesis scenario, the latent codes $\boldsymbol{c}$ can represent any continuous or discrete factor that controls the geometry of the design, *e.g.*, the upper/lower surface

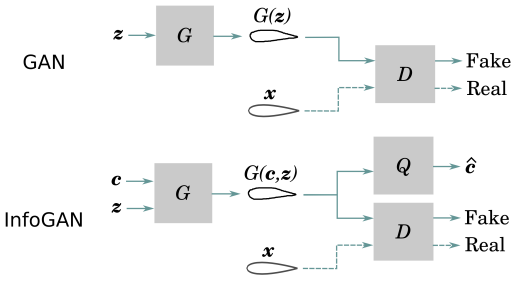


Figure 5.2: Architectures of the standard GAN and the InfoGAN

protrusion of the airfoil.

Previously there are GAN-based models that generate specific types of data (*i.e.*, images and videos) by a two-level hierarchy [159, 227]. For example, a generator first generates the structure of an image, and then conditioned on that structure, another generator generates the texture of that image. In the case of design synthesis, it is intuitive to generate each design component given all its dependencies, which results in a hierarchical model structure with multiple levels of generators.

5.3 Method

In this section, we introduce our proposed deep neural network architecture and its training details.

5.3.1 Problem Formulation

We can use a directed acyclic graph to define inter-part dependencies of a design. We call this graph a *part dependency graph*. For example, suppose we want to design an airfoil with two holes inside (top left in Fig. 5.3). We might first design the airfoil (Part A), and then set the position and diameter of one hole (Part B) based on the shape of the airfoil, followed by the design of the second hole (Part C) based on both

the airfoil shape and the first hole. Thus, the dependencies can be expressed by the graph shown in the bottom left of Fig. 5.3.

This chapter deals with a design synthesis and design space exploration problem, where we have a database of geometric designs, each of which has multiple parts. We have no prior knowledge of the specific types of constraints that confine the geometry of each part, but only inter-part dependencies. We propose a model that learns to synthesize designs based on those inter-part dependencies. We want to use this model to (1) correctly synthesize each part that follows both the shape feasibility and the dependency constraints; and (2) use a low-dimensional *latent spaces* $\mathcal{C}$ to represent the design spaces $\mathcal{X}$ of each part. The ability to learn separate representations of each part is useful for decomposing a design space exploration problem (*e.g.*, in design optimization).

Desired latent spaces satisfy the following requirements:

1. Any child latent space should be conditioned on a parent shape (*e.g.*, for the design in Fig. 5.3, any latent space of the first hole should be conditioned on an airfoil).

2. Major variability across designs in the database should be captured by those latent spaces.

3. Designs should changes consistently as we move along any basis of the latent space. This regularity/consistency/smoothness will improve latent space design exploration and optimization.

To meet the first requirement we construct a composite generative model—a model with multiple generators, each of which learns a (conditional) generative distribution of a part from the design. We ensure the rest of the requirements by adapting InfoGAN's architecture and objective, *i.e.*, conditioning the generator on latent codes

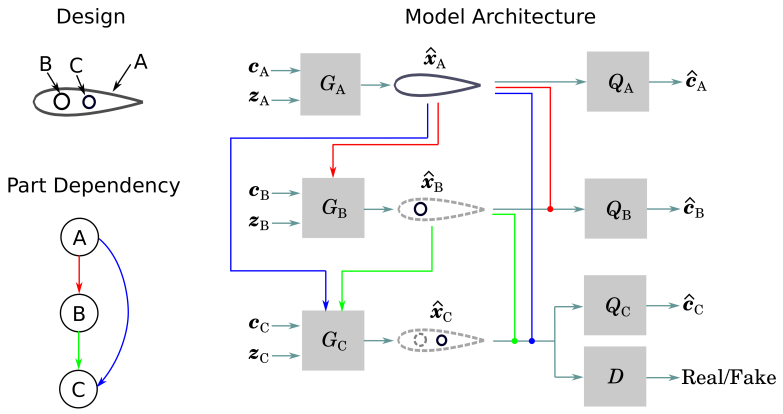


Figure 5.3: An example of part dependency and the corresponding Hierarchical GAN architecture. The interaction between generators (G_A, G_B, and G_C) and auxiliary networks (Q_A, Q_B, and Q_C) is based on the connection of the part dependency graph. This figure is best viewed in color.

and maximizing a lower bound of the mutual information between the synthesized designs and their latent representations. This is known to (1) make latent codes target salient features of the designs, and (2) disentangle the latent representation of the data space[52]. We introduce the details of our model's architecture in the following section.

5.3.2 Model Architecture

To learn separate distributions/representations of each part, we use a generative adversarial net with multiple generators/auxiliary networks. We call this network the Hierarchical Generative Adversarial Networks (HGAN). An example of its architecture is shown in Fig. 5.3. We use multiple generators to synthesize different parts in a design. The interaction between generators/auxiliary networks changes with the part dependency graph. The latent code $\boldsymbol{c}_j$ defines the latent representation of the j-th part, where $j = 1, ..., n$, and n is the number of parts. The generator G_j learns a (conditional) shape distribution $P(\boldsymbol{x}_j|\boldsymbol{c}_j, \boldsymbol{x}_{\text{Par}(j)})$, where $\text{Par}(j)$ denotes the set of

parent(s) of the j-th part. For the example in Fig. 3, $\text{Par}(A) = \emptyset$, $\text{Par}(B) = \{A\}$, and $\text{Par}(C) = \{A, B\}$. The output distribution $P_{G_j} = G_j(\boldsymbol{c}_j, \boldsymbol{z}_j, \hat{\boldsymbol{x}}_{\text{Par}(j)})$ represents the distribution of the synthesized part $\hat{\boldsymbol{x}}_j$, where $\hat{\boldsymbol{x}}_{\text{Par}(j)}$ denotes the synthesized parent part(s) of $\boldsymbol{x}_j$, and $\boldsymbol{z}_j$ is the noise input to G_j.

The auxiliary network Q_j predicts the latent code distribution of the corresponding part, *i.e.*, to estimate the conditional distributions $P(\boldsymbol{c}_j|\boldsymbol{x}_j, \boldsymbol{x}_{\text{Par}(j)})$. The discriminator D predicts whether a full design is from the database or generated by generators. A properly trained D should distinguish designs with unrealistic or mismatched parts.

The objective of HGAN is expressed as

$$\min_{\{G_j\},\{Q_j\}} \max_{D} V_{HGAN}(D, \{G_j\}, \{Q_j\}) = V(D, \{G_j\}) - \lambda L_I(\{G_j\}, \{Q_j\}) \tag{5.3}$$

where $\{G_j\}$ and $\{Q_j\}$ denotes the set of generators and auxiliary networks, respectively. The first term in Eqn. (5.3) denotes the standard GAN objective:

$$V(D, \{G_j\}) = \mathbb{E}_{\{\boldsymbol{x}_j\}\sim P_{data}}[\log D(\{\boldsymbol{x}_j\})] + \mathbb{E}_{\{\boldsymbol{c}_j\sim P(\boldsymbol{c}_j)\},\{\boldsymbol{z}_j\sim P(\boldsymbol{z}_j)\}}[\log(1 - D(\{\hat{\boldsymbol{x}}_j\}))] \tag{5.4}$$

where $\hat{\boldsymbol{x}}_j = G_j(\boldsymbol{c}_j, \boldsymbol{z}_j, \hat{\boldsymbol{x}}_{\text{Par}(j)})$. The second term in Eqn. (5.3) is the lower bound of the mutual information between the latent codes and the synthesized designs:

$$L_I(\{G_j\}, \{Q_j\}) = \sum_{j=1}^{n} \mathbb{E}_{\boldsymbol{c}_j\sim P(\boldsymbol{c}_j), \boldsymbol{x}_j\sim G_j(\boldsymbol{c}_j, \boldsymbol{z}_j, \hat{\boldsymbol{x}}_{\text{Par}(j)})}[\log Q_j(\boldsymbol{c}_j|\boldsymbol{x}_j, \hat{\boldsymbol{x}}_{\text{Par}(j)})] + H(\boldsymbol{c}_j) \tag{5.5}$$

5.4 Experimental Setup

To demonstrate the performance of our model, we built six datasets with different ground-truth inter-part dependencies. We train the proposed network on these datasets, and evaluate the generative performance, constraint satisfaction, and the

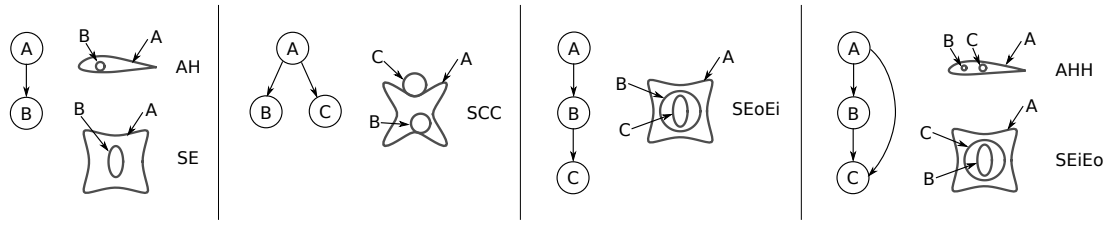


Figure 5.4: Datasets with different inter-part dependencies

latent space property through qualitative and quantitative measures.

5.4.1 Dataset

We created the following datasets, as shown in Fig. 5.4:

1. **AH**: An 2D airfoil (Part A) with a hole (circle, Part B) *inside*.

2. **AHH**: An 2D airfoil (Part A) with two *non-intersecting* holes (Parts B and C) *inside*. The centers of the two holes lie on a *horizontal* line.

3. **SE**: A superformula [79] (Part A) with a *concentric* ellipse (Part B) *inside*.

4. **SEoEi**: A superformula (Part A) with two ellipses *inside*. The second ellipse (Part C) is also *inside* the first one (Part B). All three shapes are *concentric*.

5. **SEiEo**: A superformula (Part A) with two ellipses *inside*. The second ellipse (Part C) is also *outside* the first one (Part B). All three shapes are *concentric*.[1]

6. **SCC**: A superformula (Part A) with two *tangent* circles—one (Part B) inside and the other (Part C) outside.

[1]Note that the assembly of SEiEo is the same as SEoEi, but since we synthesize its parts in a different order, the inter-part dependency changes (see Fig. 5.4). We create this dataset for comparing different inter-part dependencies, rather than simulating practical use case.

In addition, we control the part dependency graph's depth and branching factor by adding more circles/ellipses to SCC/SEoEi. This results in six extra datasets—**S**, **SC**, **SCCC**, **SCCCC**, **SEEE**, and **SEEEE**. Here the dataset of superformulas (S) is used as a baseline where no inter-part dependency is presented.

Specifically, the airfoil shapes are from the UIUC airfoil coordinates database[2], which provides the Cartesian coordinates for nearly 1,600 airfoils. Each airfoil is re-parametrized and represented with 64 Cartesian coordinates, resulting in a 64×2 matrix.

Though targeted for real-world applications, the airfoils may not be a perfect experimental dataset to visualize the latent space, because the ground truth intrinsic dimension (ID)[3] of the airfoil dataset is unknown. Thus, we create another synthetic dataset using superformulas and ellipses, the IDs of which are controllable. The superformula is a generalization of the ellipse [79]. We generate superformulas using the following equations:

$$\begin{aligned} n_1 &= 10s_1 \\ n_2 &= n_3 = 10(s_1 + s_2) \\ r(\theta) &= (|\cos\theta|^{n_2} + |\sin\theta|^{n_3})^{-\frac{1}{n_1}} \\ (x, y) &= (r(\theta)\cos\theta, r(\theta)\sin\theta) \end{aligned} \tag{5.6}$$

where $s_1, s_2 \in [0, 1]$, and (x, y) is a Cartesian coordinate. For each superformula, we sample 64 evenly spaced θ from 0 to 2π, and get 64 grid-point Cartesian coordinates. Equations (5.6) show that we can control the deformation of the superformula shape with s_1 and s_2. Thus, the ground truth ID of our superformula dataset is two.

[2] http://m-selig.ae.illinois.edu/ads/coord_database.html

[3] The intrinsic dimension is the minimum number of variables required to represent the data. It indicates the degrees of freedom we have to control the shape and position.

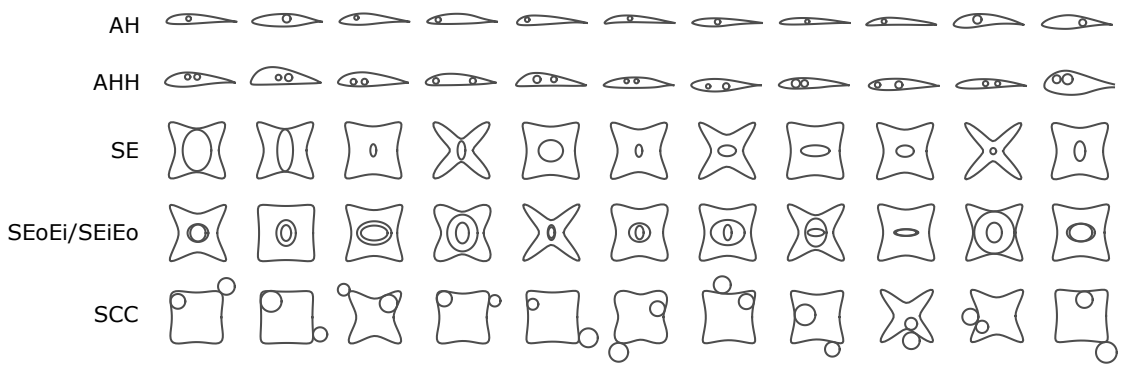


Figure 5.5: Samples drawn from datasets

For further details on how to modify the superformula parameters to adjust the ID, complexity, and number of manifolds see [49].

The other shapes (*e.g.*, circles and ellipses) are also represented using 64 Cartesian coordinates. The ground truth ID of ellipses in SE, SEoEi, and SEiEo is two, since we fix their centers and only change their semi-major axis and semi-minor axis lengths. The ground truth ID of circles in SCC is also two, since they can change their radii and move tangentially to the superformulas. The Part B (circle) in AH and AHH has a ground truth ID of three, as both their centers and radii can change; while the Part C in AHH has a ground truth ID of two, since the y-coordinate of its center is fixed. Figure 5.5 shows samples drawn from each dataset.

For each dataset, we run experiments with sample sizes ranging from 500 to 10,000.

5.4.2 Network and Training

We implement our airfoil/superformula generators by adopting the generator architecture of the BézierGAN [42, 47]. The circle/ellipse generators first generate shape parameters (*e.g.*, the center coordinates and the radius for a circle), and then convert them into grid-point coordinates using corresponding parametric functions.

For a generator with parent parts as inputs, we use an encoder to convert each parent part into a feature vector before concatenating all inputs and feeding them into the generator. Similarly, we use an encoder to convert each generated part into a feature vector before feeding them into the auxiliary networks and the discriminator. The use of encoders reduces the model complexity by reducing the input dimension of the generators, the auxiliary networks, and the discriminator. But one has to carefully choose the feature vector dimensions to avoid loss of information from dimensionality reduction.

At training, we sample the latent codes from uniform distribution $\mathcal{U}(0, 1)$, and the noise inputs from normal distribution $\mathcal{N}(0, 0.25)$. The hyper-parameter λ in Eqn. (5.3) was set to 0.1 in all experiments. The network was optimized using Adam [116] with the momentum terms $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rates were set to 0.0001. The total number of training steps was 100,000. The batch size was 32. Interested readers who wish to reproduce our exact architectures, hyper-parameters, and training procedures are directed to our code located on GitHub[4]. The training procedure is summarized in Algorithm 1.

We used TensorFlow [1] to build the networks. We trained our networks on an Nvidia Titan X GPU. For each experiment, the training process took around 2.2 hours for SE and AH, and 3 hours for other examples. The inference took less than 10 seconds.

5.5 Results and Discussion

We evaluated the performance of our trained generative models using both visual inspection (Figs. 5.6-5.10) and quantitative measures (Fig. 5.11). We analyze

[4] https://github.com/IDEALLab/hgan_jmd_2019

Algorithm 1 Train HGAN for designs with n parts

▷ T: number of training steps
▷ m: batch size
▷ X: training set
procedure TRAIN(T, m)
    for $t = 1 : T$ **do**
        Sample $\{\boldsymbol{x}^1, ..., \boldsymbol{x}^m\}$ from X, where $\boldsymbol{x}^i = [\boldsymbol{x}_1^i, ..., \boldsymbol{x}_n^i], i = 1, ..., m$
        for $j = 1 : n$ **do**
            Sample $\{\boldsymbol{c}_j^1, ..., \boldsymbol{c}_j^m\}$ from a uniform distribution
            Sample $\{\boldsymbol{z}_j^1, ..., \boldsymbol{z}_j^m\}$ from a normal distribution
        end for
        ▷ Based on Eqn. (5.3-5.5):
        Train D using $\{\boldsymbol{x}^1, ..., \boldsymbol{x}^m\}$, fixing $G_1, ..., G_n$
        $\hat{\boldsymbol{x}}^i := [G_1(\boldsymbol{c}_1^i, \boldsymbol{z}_1^i, \hat{\boldsymbol{x}}^i_{\text{Par}(1)}), ..., G_n(\boldsymbol{c}_n^i, \boldsymbol{z}_n^i, \hat{\boldsymbol{x}}^i_{\text{Par}(n)})]$
        Train $Q_1, ..., Q_n$ and D using $\{\hat{\boldsymbol{x}}^1, ..., \hat{\boldsymbol{x}}^m\}$, fixing $G_1, ..., G_n$
        Train $G_1, ..., G_n$, fixing $Q_1, ..., Q_n$ and D
    end for
end procedure

the effect of sample size and problem complexity on those quantitative performance metrics.

5.5.1 Visual Inspection.

The captured latent spaces for different examples are visualized in Figs. 5.6-5.9. All these plots are generated using a sample size of 10,000. The results show that each child latent space adjusts itself according to its parent part, so that the sizes/positions of child parts match their parent parts. This indicates that the child generator figures out the implicit constraints encoded in data. The latent spaces capture major shape variations and show consistent shape change. For example, in Fig. 5.6, the outer ellipse (middle subplot) has a consistently decreasing width from left to right, and increasing height from top to bottom. Interestingly, Fig. 5.8 shows that the circles (middle and right subplots) change in the latent space according to a polar

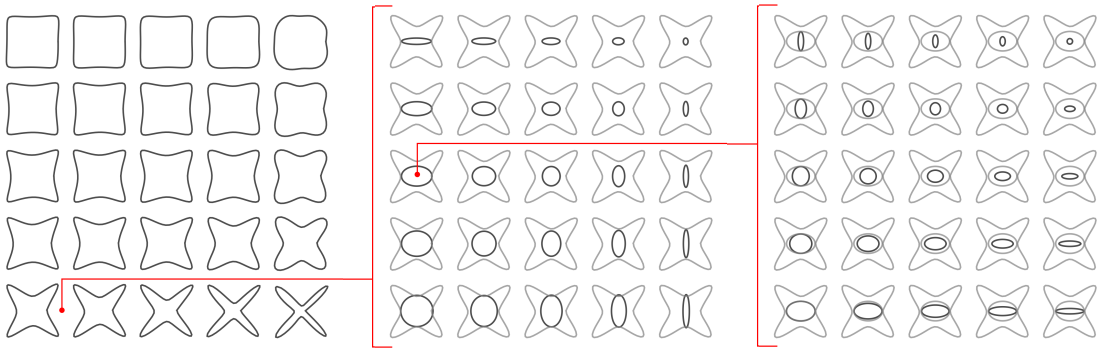


Figure 5.6: Latent space visualization of the SEoEi example. **Left**: synthesized superformulas in a 2-D latent space; **middle**: synthesized outer ellipses in a 2-D latent space conditioned on a random superformula; **right**: synthesized inner ellipses in a 2-D latent space conditioned on a random outer ellipse.

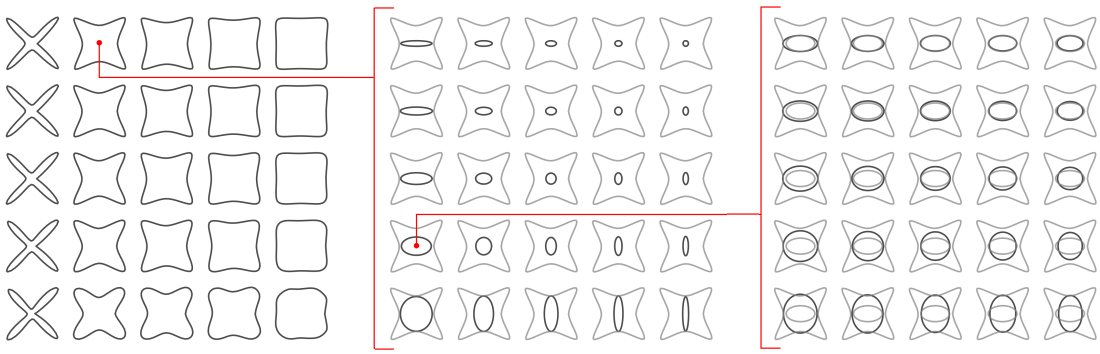


Figure 5.7: Latent space visualization of the SEiEo example. **Left**: synthesized superformulas in a 2-D latent space; **middle**: synthesized inner ellipses in a 2-D latent space conditioned on a random superformula; **right**: synthesized outer ellipses in a 2-D latent space conditioned on that same superformula and a random outer ellipse.

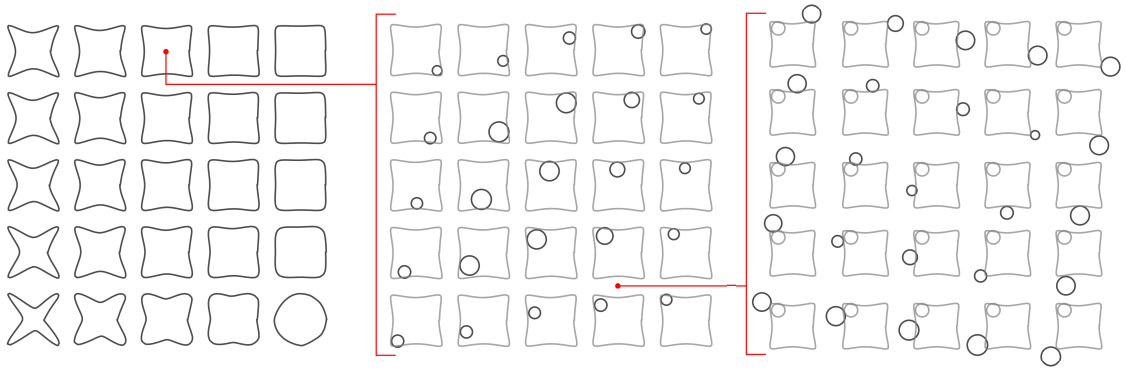


Figure 5.8: Latent space visualization of the SCC example. **Left**: synthesized superformulas in a 2-D latent space; **middle**: synthesized inner circles in a 2-D latent space conditioned on a random superformula; **right**: synthesized outer circles in a 2-D latent space conditioned on that same superformula. Interestingly, the HGAN automatically learns a polar-coordinate representation for the tangent constraint.

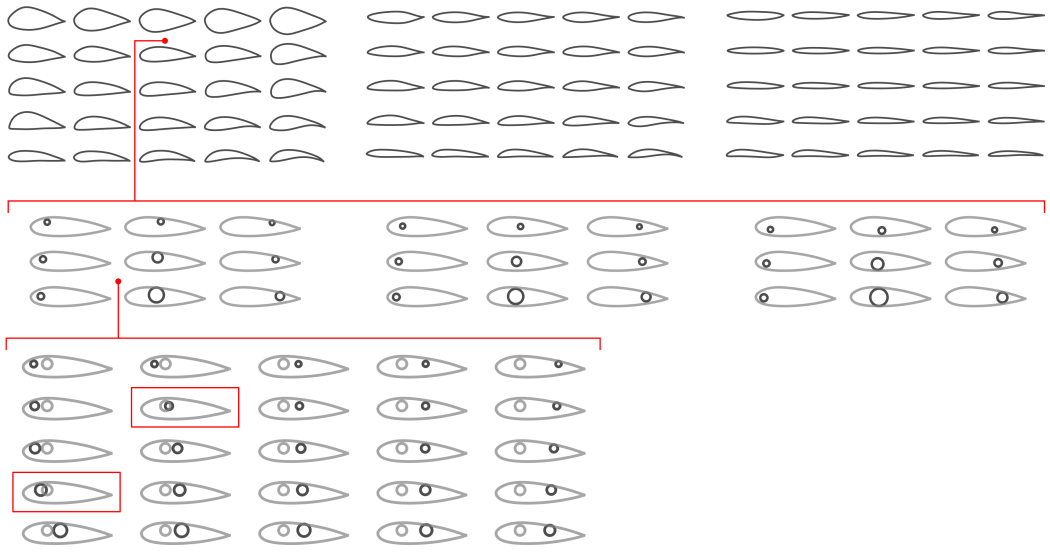


Figure 5.9: Latent space visualization of the AHH example. **Top**: Synthesized airfoils in a 3-D latent space (visualized by multiple slices of 2-D latent spaces); **middle**: Synthesized holes in a 3-D latent space conditioned on a random airfoil; **Bottom**: Synthesized holes in a 2-D latent space conditioned on that same airfoil and another random hole. Unfeasible synthesized designs occur when the two holes intersect (indicated by red boxes).

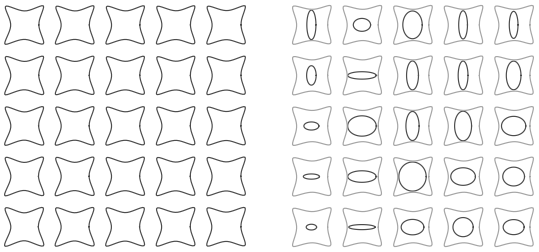


Figure 5.10: Latent spaces learned without maximizing the mutual information lower bound. **Left**: the latent code $\boldsymbol{c}$ failed to capture major shape variation; **right**: inconsistent shape variation along latent space bases (LSC=0.562 ± 0.008).

coordinate system, instead of a Cartesian coordinate system like in other examples. For example, the inner circle's radius decreases with the radial coordinate, and its position moves with the angular coordinate. This behavior is interesting because we did not explicitly encode this polar-coordinate representation into the HGAN architecture—rather, the HGAN automatically learns that such a representation is appropriate for this constraint. Figure 5.10 shows that when we removed the mutual information lower bound L_I in Eqn. (5.3), latent spaces either failed to capture major shape variation, or became entangled and inconsistent.

In the AHH example (Fig. 5.9), unfeasible synthesized designs occur when the two holes intersect. The figure shows that while the second hole moves from one side of the airfoil to the other side, it has to pass through a narrow unfeasible region. This unfeasible region cuts off the latent space, but the generator ignores this fact and learns a continuous latent space by interpolating designs inside the unfeasible region. The small volume of this unfeasible region may cause the discriminator to ignore it. In other words, the generator is willing to take the minor loss incurred by this small region of the infeasible design space in order to avoid making the latent space representation more complicated. To solve this problem, we can perform adaptive sampling in the latent space to more accurately identify the feasible region(s) [44, 45].

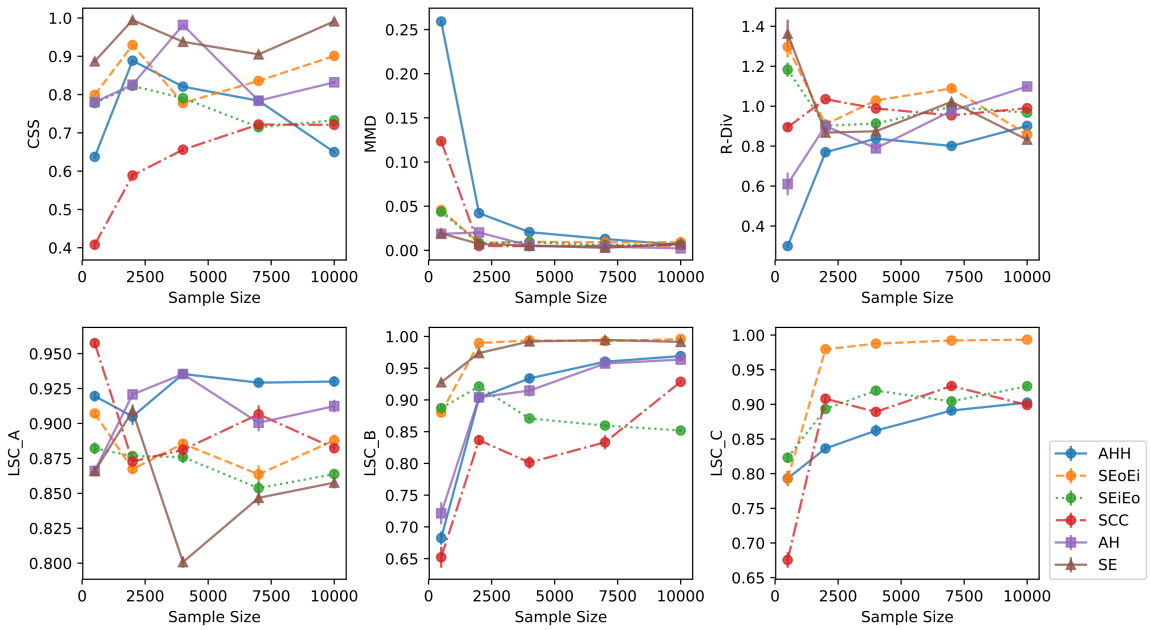


Figure 5.11: Quantitative measure of synthesized design quality and latent space properties. LSC_A, LSC_B, LSC_C denote the LSC for the latent spaces of Parts A, B, C, respectively.

5.5.2 Constraint Satisfaction.

We measure the precision of constraint satisfaction by computing the proportion of feasible designs among all the synthesized designs. We call this metric the Constraint Satisfaction Score (CSS). Specifically, different constraints define feasibility in different examples:

1. **AH**: Each point on the hole should be inside the airfoil.

2. **AHH**: (i) Each point on both holes should be inside the airfoil; (ii) the centers of the two holes should have a vertical distance of less than 0.01 (alignment constraint);[5] and (iii) the distance of the two centers should be larger than the sum of the two radii (non-intersection constraint).

[5]All designs are rescaled such that the airfoils and the superformulas have unit widths.

3. **SE**: (i) Each point on the ellipse should be inside the superformula; and (ii) The distance between the origin and the center of the ellipse should not exceed 0.01 (concentric constraint).[6]

4. **SEoEi**: (i) Each point on both ellipses should be inside the superformula; (ii) both the semi-major and semi-minor axis lengths of the first ellipse should be larger than the second one; and (iii) for both ellipses, the distance between their centers and the origin should not exceed 0.01 (concentric constraint).

5. **SEiEo**: Same as the SEoEi example, except that the first and the second ellipses are swapped.

6. **SCC**: For each circle with a center C_o and a radius r, the difference between r and the distance from C_o to the superformula should be less than 0.03 (tangent constraint).

The results on CSS in Fig. 5.11 show that the tangent constraint in the example SCC is the hardest to learn. But the learning performance improves with larger sample size. The SE example outperforms SEoEi and SEiEo on CSS, which is expected since the task of learning constraints in SE can be considered as a sub-task in SEoEi and SEiEo. This also applies to AH and AHH. It is also expected that SEoEi outperforms SEiEo, since the two are dealing with the same design but the former has fewer dependencies.

5.5.3 Distance between Data and Generator Distributions.

We measure how well our generator approximates the real data distribution by computing the kernel maximum mean discrepancy (MMD) [87] between the data and

[6]For all the examples we assume that the superformula is centered at the origin.

the generator distribution:

$$\text{MMD}^2(P_{data}, P_G) \quad = \quad \mathbb{E}_{\boldsymbol{x}_d, \boldsymbol{x}'_d \sim P_{data}; \boldsymbol{x}_g, \boldsymbol{x}'_g \sim P_G} \left[k(\boldsymbol{x}_d, \boldsymbol{x}'_d) - 2k(\boldsymbol{x}_d, \boldsymbol{x}_g) + k(\boldsymbol{x}_g, \boldsymbol{x}'_g) \right]$$

where $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/(2\sigma^2)\right)$. A lower kernel MMD indicates that the generator distribution is closer to the data distribution.

The top middle plot in Fig. 5.11 shows the results of MMD. Here the SEoEi and SEiEo examples have similar MMD values, which suggests that synthesized designs in both examples have similar perceptual quality, despite the fact that SEiEo requires more model parameters to learn the additional dependency. The MMD plot provides insight into how realistic the generated designs can get as the training sample size changes. In general, MMD first decreases steeply with the sample size, and then reaches a plateau at some point. This point indicates the smallest sample size required to reach a "perceptually good" synthesis performance.

5.5.4 Diversity of Generated Designs.

A common problem in GANs' training is mode collapse, during which the generator only generates a few types of designs to fool the discriminator instead of properly learning the complete data distribution. Therefore it is important to measure the diversity of the synthesized designs. We use Relative Diversity (R-Div) to measure the relative level of variability captured by the generator.

We define the diversity of N samples $\boldsymbol{X} \in \mathbb{R}^{d \times N}$ as[7]

$$\text{Div}(\boldsymbol{X}) = \frac{1}{N}\text{trace}(\text{cov}[\boldsymbol{X}, \boldsymbol{X}])$$

[7]Here each sample is represented as a column vector containing all the coordinates of the design.

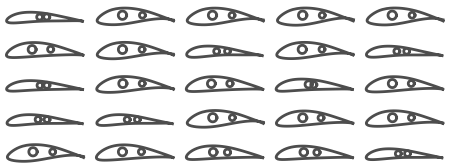


Figure 5.12: Mode collapse occurs when the sample size is 500. The diversity is low in these randomly generated designs.

The R-Div metric can then be expressed as

$$\text{R-Div} = \frac{\text{Div}(\boldsymbol{X}_g)}{\text{Div}(\boldsymbol{X}_{data})}$$

where $\boldsymbol{X}_g$ and $\boldsymbol{X}_{data}$ denote the set of synthesized designs and designs from the dataset, respectively. A R-Div close to 0.0 means that there is little variation within the synthesized designs, which could be an indicator of mode collapse. A R-Div around 1.0 indicates that the synthesized designs have a similar level of variability with the dataset. Note that high diversity does not always indicate good performance, as there could be unrealistic designs being synthesized, which also contribute to diversity. Thus, we should view this metric in concert with the kernel MMD to determine how well the generator performs.

Figure 5.11 shows that synthesized designs tend to have a high divergence of R-Div when the sample size is small. Particularly, in the AHH example with a sample size of 500, the low R-Div combined with the high kernel MMD indicates the occurrence of mode collapse (Fig. 5.12). Increasing the sample size stabilized the R-Div and eventually bounded it between 0.8 and 1.1.

5.5.5 Latent Space Consistency.

A desirable latent space has two properties: 1) *disentanglement*: each latent variable is related to only one factor; and 2) *consistency*: shapes change consistently along any basis of the latent space. Note that this consistency is evaluated along one direction at a time, since scales along different directions may vary. To the best of our knowledge, existing quantitative measurements for the first property—latent space disentanglement—are supervised, *i.e.*, the ground-truth independent factors causing shape deformation have to be provided [39, 95, 114]. The second property is important for latent space design exploration. When searching for designs along a direction in the latent space, optimization algorithms and humans usually prefer if shapes change consistently, such that the objective function over the latent space is less complicated (*i.e.*, has better Lipschitz continuity) and has fewer local optima.

We propose Latent Space Consistency (LSC) as a quantitative measure of how consistently shapes change along any basis of the latent space. Since the change from one shape to another can be measured by their dissimilarity, distances between samples along a certain direction in the latent space should be consistent with the dissimilarity between those samples. We use Pearson correlation coefficient to measure this consistency. Algorithm 2 describes how to compute the LSC. The choice of the dissimilarity function d is not central to the overall method. In our experiments, we simply use the Euclidean distance to measure the dissimilarity of designs.

Both the middle plots in Figs. 5.6 and 5.7 show latent spaces with LSCs above 0.9. In contrast, the right plot in Fig. 5.10 provides a visual example of an LSC of around 0.56. The bottom plots in Fig. 5.11 shows that larger sample size does not improve LSCs of Part A (at least with sample sizes in the range from 500 to 10,000), but improves the LSCs of Parts B and C in most cases.

Algorithm 2 Evaluate Latent Space Consistency

procedure LATENTCONSISTENCY(G, m, n, d)
    ▷ G: the mapping from a latent space to a design space
    ▷ m: the number of lines to be evaluated
    ▷ n: the number of points sampled on each line
    ▷ d: a dissimilarity function
    ▷ $\mathcal{C}$: the latent space
    ▷ $\mathcal{X}$: the design space
    $sum = 0$
    for $i = 1 : m$ **do**
        Sample a line $\mathcal{L}$ parallel to any basis of $\mathcal{C}$
        Sample n points $\{\boldsymbol{c}^1, \boldsymbol{c}^2, ..., \boldsymbol{c}^n\}$ along $\mathcal{L}$
        $\{\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^n\} := \{G(\boldsymbol{c}^1), G(\boldsymbol{c}^2), ..., G(\boldsymbol{c}^n)\}$
        $D_{\mathcal{C}} := \{\|\boldsymbol{c}^i - \boldsymbol{c}^j\|\}$, $D_{\mathcal{X}} := \{d(\boldsymbol{x}^i, \boldsymbol{x}^j)\}$, where $i, j \in \{1, ..., n\}$
        Compute Pearson correlation coefficient:

$$\rho := \frac{\mathrm{cov}(D_{\mathcal{C}}, D_{\mathcal{X}})}{\sigma(D_{\mathcal{C}})\sigma(D_{\mathcal{X}})}$$

        $sum := sum + \rho$
    end for
    Return sum/m
end procedure

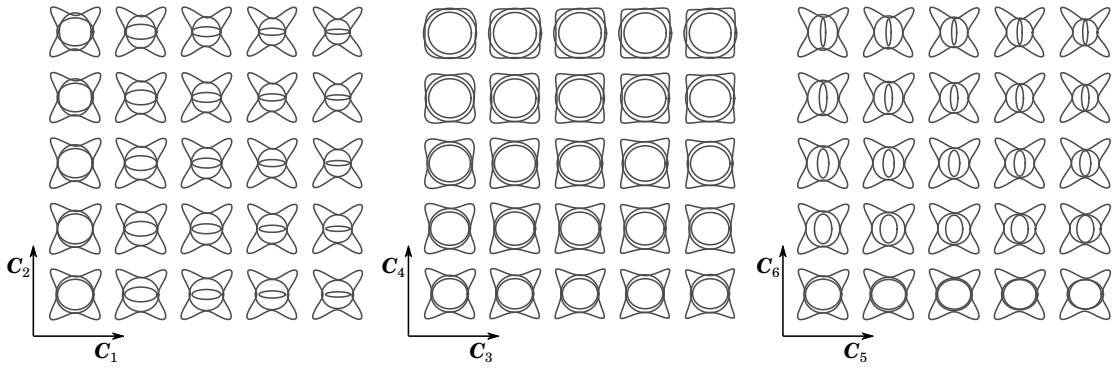


Figure 5.13: The latent space learned by a standard InfoGAN. This plot visualizes a 6-D latent space of the SEoEi example by showing each two dimensions while setting the latent coordinates of other dimensions to zero. In each dimension, three parts change simultaneously. This shows that the latent space learned by a standard InfoGAN does not disentangle each part's shape variation. In contrast, the results of the HGAN disentangle and separate each part's latent space.

5.5.6 Effect of Encoding Inter-Part Dependencies.

To further study the effect of encoding inter-part dependencies in the GAN, we compared the results of our HGAN with a standard InfoGAN, where there is only one generator and all parts are synthesized from a single latent space. The latent space dimension was set to the sum of all latent space dimensions in the HGAN experiments. Figure 5.13 visualizes the latent space learned by a standard InfoGAN. It shows that in each latent dimension, three parts change simultaneously, which indicates that the latent space does not disentangle each part's shape variation.

We also study the effects of part dependency graph's depth and branching factor on the model's synthesis performance. The examples S, SE, SEE, SEEE, and SEEEE simulate increasing depths, respectively; while the examples S, SC, SCC, SCCC, and SCCCC simulate increasing branch factors, respectively. The results are shown in Fig. 5.14. The InfoGAN model is used as a baseline where we do not encode inter-part dependencies. As expected for both HGAN and InfoGAN, CSS decreases

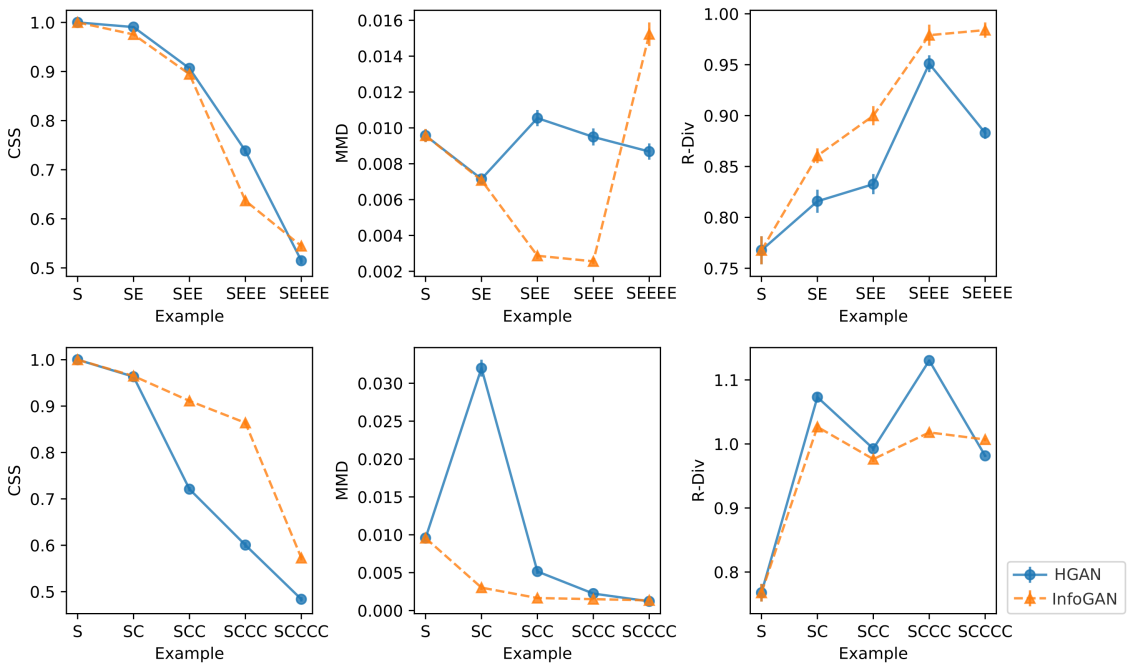


Figure 5.14: Effects of depth (top) and branching factor (bottom) on synthesis performance. Here we denote the example SEoEi as SEE for simplicity.

with an increased number of parts, since additional parts bring extra constraints. Theoretically, encoding inter-part dependencies introduces extra constraints to the model and hence complicates the overall task. However, based on the results, the HGAN shows no significant performance drop comparing to the InfoGAN, except for the CSS when increasing the branching factor and the R-Div when increasing the depth. Note that the MMD values are below 0.05 and do not change notably. The lower CSS or R-Div indicates that HGAN compromises its synthesis precision (*i.e.*, the precision of satisfying constraints) or generator distribution's coverage for disentangling each part's latent space. We also included the training history of HGAN and InfoGAN in the online supplemental material.

5.6 Limitations and Future Work

One limitation of our approach is that it is difficult to achieve high precision in satisfying some constraints. This problem exists in all purely data-driven design methods, where there is no process of incorporating restricted constraints in the generative model or validating the constraint satisfaction of the outputs. While we can address this by encoding constraints explicitly in the generative model, this requires us to know these constraints in advance and create hand-coded rules for all types of constraints. For example, to generate concentric ellipses, one can simply fix the center of the second ellipse to have the same coordinate as the first one. However, it may be difficult to incorporate some constraints in the generative model (*e.g.*, the constraint of one part being inside another). Also, it is sometimes hard to even know the type of the exact constraint between parts (*e.g.*, aesthetic preferences when placing handles on a vase). Thus, here we assume that we have no prior knowledge of the types of constraints, and inter-part dependencies are the only knowledge we need for our model. Despite the limitation of purely data-driven design methods, they can be used in the conceptual design stage for exploring a wide range of design alternatives and inspiring novel designs. We can also use validation based on simulation, experiment, or human annotation to exclude infeasible synthesized designs when performing latent space exploration [44, 89], which could be an interesting avenue for future work.

Another limitation is that all designs must have the same part dependency graph, which is impractical in some cases. For example, not all tables have four legs, thus for some designs, their part dependency graphs might miss some nodes. Future study needs to address this situation.

Sometimes the design data is not partitioned into separate components. One possible solution to this problem is to apply unsupervised co-segmentation [194] to

partition designs and establish correspondences between common components in different designs. Then we can use HGAN to learn the latent spaces and generate new designs.

5.7 Summary

We introduced a GAN-based generative model for synthesizing designs with inter-part dependencies. It decomposes the synthesis into synthesizing each part conditioned on the corresponding parent part. This also creates a conditioned low-dimensional latent representation that allows accelerated design exploration. This model is built for problems where design space exploration over the latent space has to be staged since the optimal solution of one part depends on the geometry of another. Such models can accelerate design optimization problems, which we are exploring in our future work.

An advantage of neural-network-based generative models (*e.g.*, GANs and VAEs), compared to other dimensionality reduction models (*e.g.*, PCA and GPLVM), is that one can define or regularize latent distributions. Our model adapts InfoGAN's mutual information objective to derive a consistent latent space, where the change of shapes is consistent along any basis of the latent space. This property is desirable in latent space design exploration, as the objective function over the latent space is less complicated and has less local optima.

We also created new benchmark datasets for studying different kinds of inter-part dependencies including: type of geometric constraints, depth of hierarchy, and branching factor of parent/child relationships. By using these datasets, we characterized the effects of sample size and part dependencies' complexity (depth and branching factor) on the synthesis performance of our generative model. We also proposed a new eval-

uation metric for generative models that measures the consistency of shape variation in the latent space. Compared to a standard InfoGAN, the HGAN disentangles each part's latent space at the cost of weakened synthesis precision when the branching of the part dependency increases.

The concept of decoupling the latent space of a design is important for design space exploration in general. It allows separate exploration and synthesis of each part, and helps us understand how different constraints control shape variation. Though we use GANs to achieve this goal, the idea of encoding inter-part dependencies, learning conditioned generative distributions, and the latent space analysis is also applicable to other generative models. Thus this chapter's techniques lay the foundation for learned generative models to extend to more realistic engineering systems where inter-part dependencies are widespread.

So far this dissertation has shown how to measure the intrinsic complexity of a design space to guide data-driven design synthesis and how to incorporate prior knowledge into data-driven design synthesis models. The compact latent spaces learned by the proposed models can reduce the cost of design space exploration. But there is another problem in design space exploration—the bounds of the design space are hard to specify. Usually one cannot guarantee that the desired designs are inside the user-specified bounds. Also sometimes the desired design is remarkably different from existing ones, which makes it unreliable to set bounds based on the boundary of any design dataset. To address this problem, the next two chapters will introduce design space exploration methods that gradually expand an input space (*i.e.*, either a design space or a latent space), so that no fixed bounds are required.

Chapter 6: Feasible Domain Identification with Active Expansion Sampling

Portions of the work in this chapter were published as journal papers in the Journal of Mechanical Design [44] and Structural and Multidisciplinary Optimization [43]

6.1 Introduction

In applications like design space exploration [64, 128, 237] and reliability analysis [133, 248], people need to find feasible domains within which solutions are valid. Sometimes the constraints that define those feasible domains are implicit, *i.e.*, they cannot be represented analytically. Examples of these constraints are aesthetics, functionality, or performance requirements, which are usually evaluated by human assessment, experiments, or time-consuming computer simulations. Thus usually it is expensive to detect the feasibility of a given input. In such cases, one would like to use as few samples as possible while still approximating the feasible domain well.

To solve such problems, researchers have used *active learning* (or *adaptive sampling*)[1] to sequentially select the most informative instances and query their feasibility, so that the number of queries can be minimized [101, 128, 133, 175, 248]. These methods require fixed bounds over the input space, and only pick queries inside those bounds. But what if we do not know how wide to set those bounds? If we set the

[1]Note that in this chapter the terms "active learning" and "adaptive sampling" are interchangeable.

bounds too large, an active learner will require an excessively large budget to explore the input space; whereas if we set the bounds too small, we cannot guarantee that an algorithm will recover all the feasible domains [44]. In this case, we need an active learning method that can gradually expand our knowledge about the input space until we have either discovered all feasible domains or used up our remaining query budget.

This chapter proposes a method — which we call *Active Expansion Sampling* (AES) — to solve that problem by casting the detection of feasible domains as an *unbounded domain estimation problem*. In an unbounded domain estimation problem, given an expensive function $h : \mathcal{X} \in \mathbb{R}^d \to \{-1, 1\}$ that evaluates any point $\boldsymbol{x}$ in an *unbounded* input data space $\mathcal{X}$, we want to find (possibly disconnected) feasible domains in which $h(\boldsymbol{x}) = 1$. Specifically, h could be costly computation, time-consuming experiments, or human evaluation, so that the problem cannot be solved analytically. By *unbounded*, we mean that we do not manually bound the input space. Thus the input space can be considered as infinite, and theoretically if the query budget allows, our method can keep expanding the explored area of the input space. To use as few function evaluations as necessary to identify feasible domains, AES first fully exploits (up to an accuracy threshold) any feasible domains it knows about and then, budget permitting, searches outward to discover other feasible domains.

The main contributions of this chapter are:

1. We introduce the AES method for identifying (possibly disconnected) feasible domains over an unbounded input space.

2. We provide a framework that transfers bounded active learning methods into methods that can operate over unbounded input space.

3. We introduce a dynamic local pool method that efficiently finds near-optimal

solutions to the global optimization problem (Eq. 6.9) for selecting queries.

4. We prove a constant theoretical bound for AES's misclassification error at any iteration inside the explored region.

6.2 Related Work

Essentially, the unbounded domain estimation problem breaks down into two tasks explored by past researchers: 1) the active learning task, where we efficiently query the feasibility of inputs; and 2) the classification task, where we estimate decision boundaries (*i.e.*, boundaries of feasible domains) that separates the feasible class and the infeasible class (*i.e.*, feasible regions and infeasible regions). For the first task, we will review relevant past work on active learning. For the second task, we use the Gaussian Process as the classifier in this chapter and will introduce basic concepts of Gaussian Processes.

6.2.1 Feasible Domain Identification

Past work in design and optimization has proposed ways to identify feasible domains or decision boundaries of expensive functions. Generally, those methods were proposed to reduce the number of simulation runs and improve the accuracy of surrogate models in simulation-based design and reliability assessment [13, 133]. Also, the problem of feasible domain identification is equivalent to estimating the level set or the threshold boundaries of a function, where the feasible/infeasible region becomes superlevel/sublevel set [26, 86]. Such methods select samples that are expected to best improve the surrogate model's accuracy. A common rule is to sample on the estimated decision boundary, but not close to existing sample points. Existing methods achieve this by (1) explicitly optimizing or constraining the decision function or the

distance between the new sample and the existing samples [12, 13, 195], or (2) selecting points based on the estimated function values and their confidence at candidate points [26, 53, 54, 86, 133, 235].

6.2.2 Active Learning

Methods for feasible domain identification usually require strategies that sequentially sample points in an input space, such that the sample size is minimized. These strategies fall under the larger category of active learning.

There are three main scenarios of active learning problems: (1) membership query synthesis, (2) stream-based selective sampling, and (3) pool-based sampling [187]. In the *membership query model*, the learner generates samples de novo for labeling. For classification tasks, researchers have typically applied membership query models to learning finite concept classes [6, 9, 103, 115] and halfspaces [3, 38]. In the *stream-based selective sampling model*, an algorithm draws each unlabeled sample from an incoming data distribution, and then decides whether or not to query that label. This decision can be based on some informativeness measure of the drawn sample [2, 33, 34, 58, 61, 75, 160, 181], or whether the drawn sample is inside a *region of uncertainty* [57, 60]. In the *pool-based sampling model*, there is a small pool of labeled samples and a large (but finite) pool of unlabeled samples, where the learner selects new queries from the unlabeled pool.

The unbounded domain estimation problem assumes that synthesizing an unlabeled sample from the input space is not expensive (as in the membership query scenario), since otherwise we have to use existing samples and the input space will be bounded. An example that satisfies this assumption is experimental design, where we can form an experiment by selecting a set of parameters. With this assumption,

our proposed method approximates the pool-based sampling setting by synthesizing a pool of unlabeled samples in each iteration.

A pool-based sampling method first trains a classifier using the labeled samples. Then it ranks the unlabeled samples based on their *informativeness* indicated by an *acquisition function*. A query is then selected from the pool of unlabeled samples according to their rankings. After that, we add the selected query into the set of labeled data and repeat the previous process until our query budget is reached. Many of these methods use the informativeness criteria that select queries with the maximum label ambiguity [100, 136, 188], contributing the highest estimated expected classification error [31, 125, 154, 247], best reducing the *version space* [215], or where different classifiers disagree the most [7, 148]. Such methods are usually good at *exploitation*, since they keep querying points close to the decision boundary, refining our estimate of it.

However, when the input space may have multiple regions of interest (*i.e.*, feasible regions), these methods may not work well if the active learner is not aware of all the regions of interest initially. Note that while some of the methods mentioned above also consider representativeness [100, 148, 154, 188, 247], or the diversity of queries [97, 236], they do not explicitly explore unknown regions and discover other regions of interests. To address this issue, an active learner also has to allow for *exploration* (*i.e.*, to query in unexplored regions where no labeled sample has been seen yet). A learner must trade-off exploitation and exploration.

To query in an unexplored region, there are methods that (1) take into account the predictive variance at unlabeled samples when selecting new queries [26, 86, 111], (2) naturally balance exploitation/exploration by looking a the expected error [143], or (3) make exploitative and exploratory queries separately using different strategies [10, 22, 96, 98, 124, 161]. In previous methods, the exploitation-exploration trade-off

was performed in a bounded input space or a fixed sampling pool. However, in the unbounded domain estimation problem, there is no fixed sampling pool and we are usually uncertain about how to set the bounds of the input space for performing active learning. If the bounds are too small, we might miss feasible domains; while if the bounds are too large, the active learner has to query more samples than necessary to achieve the required accuracy.

In this chapter, we introduce a method of using active learning to expand our knowledge about an unbounded input data space, and discover feasible domains in that space. A naïve solution would be to progressively expand a bounded input space, and apply the existing active learning techniques. However, there are two problems with this naïve solution: (1) it is difficult to explicitly specify when and how fast we expand the input space; and (2) the area we need to evaluate increases over time increasing the computational cost. Thus existing active learning techniques cannot apply directly to the unbounded domain estimation problem. To the best of our knowledge, [44] is the first to deal with the active learning problem over an unbounded input space (*i.e.*, the unbounded domain estimation problem). The AES method proposed in this chapter improves upon that previous work (as illustrated in Sect. 6.3).

6.2.3 Gaussian Process Classification (GPC)

Gaussian Processes (GP, also called Kriging) are often used as a classifier in active learning [26, 53, 54, 86, 111, 133]. Compared to other commonly used classifiers such as Support Vector Machines or Logistic Regression, GP naturally models probabilistic predictions. This offers us a way to evaluate a sample's informativeness based on its predictive probability distribution.

The Gaussian process uses a kernel (covariance) function $k(\boldsymbol{x}, \boldsymbol{x}')$ to measure the similarity between the two points $\boldsymbol{x}$ and $\boldsymbol{x}'$. It encodes the assumption that "similar inputs should have similar outputs". Some commonly used kernels are the Gaussian kernel and the exponential kernel [110, 124, 142, 143]. In this chapter we use the Gaussian kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2l^2}\right) \tag{6.1}$$

where l is the length scale.

For binary GP classification, we place a GP prior over the latent function $f(\boldsymbol{x})$, and then "squash" $f(\boldsymbol{x})$ through the logistic function to obtain a prior on $\pi(\boldsymbol{x}) = \sigma(f(\boldsymbol{x})) = P(y = 1|\boldsymbol{x})$. In the feasible domain identification setting, we can consider $f : \mathcal{X} \in \mathbb{R}^d \to \mathbb{R}$ as an estimation of feasibility, thus we can call it *estimated feasibility function*. Under the Laplace approximation, given the labeled data $(X_L, \boldsymbol{y})$, the posterior of the latent function $f(\boldsymbol{x})$ at any $\boldsymbol{x} \in X_U$ is a Gaussian distribution: $f(\boldsymbol{x})|X_L, \boldsymbol{y}, \boldsymbol{x} \sim \mathcal{N}(\bar{f}(\boldsymbol{x}), V(\boldsymbol{x}))$ with the mean and the variance expressed as

$$\bar{f}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T K^{-1} \hat{\boldsymbol{f}} = \boldsymbol{k}(\boldsymbol{x})^T \nabla \log P(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{6.2}$$

$$V(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^T (K + W^{-1})^{-1} \boldsymbol{k}(\boldsymbol{x}) \tag{6.3}$$

where $W = -\nabla\nabla \log P(\boldsymbol{y}|\boldsymbol{f})$ is a diagonal matrix with non-negative diagonal elements; $\boldsymbol{f}$ is the vector of latent function values at X_L, *i.e.*, $f_i = f(\boldsymbol{x}^{(i)})$ where $\boldsymbol{x}^{(i)} \in X_L$; K is the covariance matrix of the training samples, *i.e.*, $K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$; $\boldsymbol{k}(\boldsymbol{x})$ is the vector of covariances between $\boldsymbol{x}$ and the training samples, *i.e.*, $k_i(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$; and $\hat{\boldsymbol{f}} = \arg\max_{\boldsymbol{f}} P(\boldsymbol{f}|X, \boldsymbol{y})$. When using the Gaussian kernel shown in Eq. 6.1, $k(\boldsymbol{x}, \boldsymbol{x}) = 1$. We refer interested readers to a detailed description by Rasmussen [172] about the Laplace approximation for the binary GP classifier.

The decision boundary corresponds to $\bar{f}(\boldsymbol{x}) = 0$ or $\bar{\pi}(\boldsymbol{x}) = 0.5$. We predict $y = -1$ when $\bar{f}(\boldsymbol{x}) < 0$, and $y = 1$ otherwise.

6.3 Active Expansion Sampling (AES)

Algorithm 3 summarizes our proposed Active Expansion Sampling method. Overall, the method consists of the following steps:

1. Select an initial sample $\boldsymbol{x}^{(0)}$ to label.
2. In each subsequent iteration,
   (a) check the exploitation/exploration status (Sect. 6.4.4),
   (b) generate a pool of candidate samples X_U based on the exploitation/exploration status and previous queries (Sect. 6.4.2 and 6.4.3),
   (c) train a GP classifier using the labeled set X_L to evaluate the informativeness of candidate samples in X_U,
   (d) select a sample from X_U based on its informativeness and its distance from $\boldsymbol{c}$ (Sect. 6.3.1),
   (e) label the new sample and put it into X_L.
3. Exit when the query budget is reached.

This AES method improves upon our previous domain expansion method [44] in several ways. For example, the previous method generates a pool X_U that expands with the explored region each iteration. So its pool size and hence the computational cost increase significantly over time if using a constant sample density. To avoid this problem, this chapter proposes a dynamic local pool method (Sect. 6.4). Another

Algorithm 3 The Active Expansion Sampling algorithm

Inputs:
  Query budget T
  Initial point $\boldsymbol{x}^{(0)}$ and its label y_0
  d-dimensional evaluation function $h(\cdot)$
  Hyperparameters ϵ and τ
Initialize:
  $X_L \leftarrow \{\boldsymbol{x}^{(0)}\}$, $Y_L \leftarrow \{y_0\}$, $INIT \leftarrow True$
for $t = 1, 2, \ldots, T$ **do**
  if $INIT$ is $True$ **then**
    if X_L consists of only one class (all feasible or all infeasible) **then**
      $\boldsymbol{c} \leftarrow \boldsymbol{x}^{(0)}$
    else
      $INIT \leftarrow False$
      $\boldsymbol{c} \leftarrow$ centroid of positive samples in X_L
    end if
  end if
  Train the GP classifier using X_L
  Compute $\delta_{exploit}$ using Eq. 6.18
  $X_U \leftarrow$ uniform samples inside the $(d-1)$-sphere $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{exploit})$
  Compute $\bar{f}(\boldsymbol{x})$, $V(\boldsymbol{x})$, and $p_\epsilon(\boldsymbol{x})$ for $\boldsymbol{x} \in X_U$ using Eq. 6.2, (6.3), and (6.4)
  if there are both $\bar{f}(\boldsymbol{x}) < 0$ and $\bar{f}(\boldsymbol{x}) > 0$ for $\{\boldsymbol{x} \in X_U | p_\epsilon(\boldsymbol{x}) > \tau\}$ **then** ▷ Exploitation stage
    Select a new query $\boldsymbol{x}^{(t)}$ from X_U based on Eq. 6.9
  else ▷ Exploration stage
    Compute $\delta_{explore}$ using Eq. 6.17
    if previous iteration is in exploitation stage **then**
      $\hat{\boldsymbol{x}} \leftarrow \text{argmax}_{\boldsymbol{x} \in X_L} \|\boldsymbol{x} - \boldsymbol{c}\|$
      $X_U \leftarrow$ uniform samples inside the $(d-1)$-sphere $\mathcal{C}(\hat{\boldsymbol{x}}, \delta_{explore})$
    else
      $X_U \leftarrow$ uniform samples inside the $(d-1)$-sphere $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{explore})$
    end if
    Compute $\bar{f}(\boldsymbol{x})$, $V(\boldsymbol{x})$, and $p_\epsilon(\boldsymbol{x})$ for $\boldsymbol{x} \in X_U$ using Eq. 6.2, 6.3, and 6.4
    Select a new query $\boldsymbol{x}^{(t)}$ from X_U based on Eq. 6.9
  end if
  $y_t \leftarrow h(\boldsymbol{x}^{(t)})$
  $X_L \leftarrow X_L \cup \{\boldsymbol{x}^{(t)}\}$, $Y_L \leftarrow Y_L \cup \{y_t\}$
end for

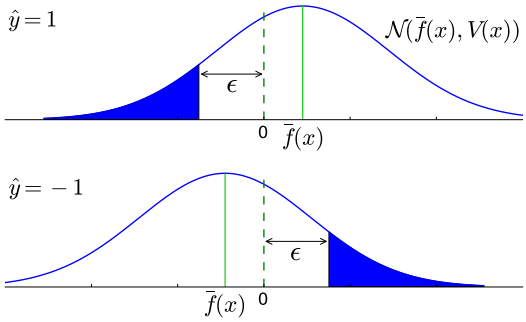


Figure 6.1: The probability density function of the latent function $f(\boldsymbol{x})$ [44]. The blue areas represent the ϵ-margin probability $p_\epsilon(\boldsymbol{x})$.

major difference is that AES provides a verifiable way to distinguish between exploitation and exploration (Sect. 6.4.4); while the previous method uses a heuristic based on the labels of last few queries (which is more likely to make mistakes). In this section and Sect. 6.7, we show comprehensive theoretical analysis and experiments to prove favorable properties of our new method.

6.3.1 ϵ-Margin Probability

We train a GP classification model to evaluate the informativeness of candidate samples based on the *ϵ-margin probability* (Fig. 6.1):

$$
\begin{aligned}
p_\epsilon(\boldsymbol{x}) &= \begin{cases} P(f(\boldsymbol{x}) < -\epsilon|\boldsymbol{x}), & \text{if } \hat{y}=1 \\ P(f(\boldsymbol{x}) > \epsilon|\boldsymbol{x}), & \text{if } \hat{y}=-1 \end{cases} \\
&= P(-\hat{y}f(\boldsymbol{x}) > \epsilon|\boldsymbol{x}) \\
&= P(\hat{y}f(\boldsymbol{x}) < -\epsilon|\boldsymbol{x}) \\
&= \Phi\left(-\frac{|\bar{f}(\boldsymbol{x})|+\epsilon}{\sqrt{V(\boldsymbol{x})}}\right)
\end{aligned}
\tag{6.4}
$$

where $\hat{y}$ is the estimated label of $\boldsymbol{x}$, the margin $\epsilon > 0$, and $\Phi(\cdot)$ is the cumulative distribution function of standard Gaussian distribution $\mathcal{N}(0, 1)$. The ϵ-margin probability represents the probability of $\boldsymbol{x}$ being misclassified with some degree of certainty (controlled by the margin ϵ). Let the misclassification loss be

$$L(\boldsymbol{x}) = \begin{cases} \max\{0, -f(\boldsymbol{x})\}, & \text{if } y = 1 \\ \max\{0, f(\boldsymbol{x})\}, & \text{if } y = -1 \end{cases} \tag{6.5}$$
$$= \max\{0, -yf(\boldsymbol{x})\}$$

where y is the true label of $\boldsymbol{x}$. $L(\boldsymbol{x})$ measures the deviation of the estimated feasibility function value $f(\boldsymbol{x})$ from 0 when the class prediction is wrong. Then, based on Eq. 6.4 and 6.5, $p_\epsilon(\boldsymbol{x}) = P(L(\boldsymbol{x}) > \epsilon)$, which is the probability that the expected misclassification loss exceeds ϵ. A high $p_\epsilon(\boldsymbol{x})$ indicates that $\boldsymbol{x}$ is very likely to be misclassified, and requires further evaluation. Thus we use this probability to measure informativeness.

6.3.2 Exploitation and Exploration

Since our input space is unbounded, naïvely maximizing the ϵ-margin probability (informativeness) will always query points infinitely far away from previous queries.[2] To avoid this issue, one solution is to query informative samples that are close to previously labeled samples. This allows the active learner to progressively expand its knowledge as the queries cover an increasingly large area of the input space. When a new decision boundary is discovered during expansion, we want a query strategy that continues querying points on that decision boundary, such that the new feasible region

[2] A point infinitely far away from previous queries has the $\bar{f}(\boldsymbol{x})$ close to 0 and the maximum $V(\boldsymbol{x})$, thus the highest $p_\epsilon(\boldsymbol{x})$.

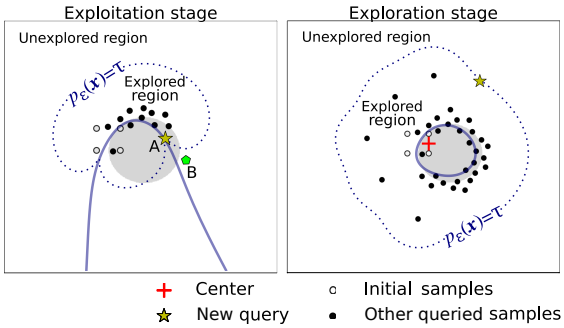


Figure 6.2: Queries at the exploitation stage (left) and the exploration stage (right). The gray area is the ground truth of the feasible domain. The solid line is the decision boundary estimated by the GP classifier, and the dotted line is the isocontour of $p_\epsilon(\boldsymbol{x})$. At the exploitation stage (left), the center $\boldsymbol{c}$ is the previous query, which makes the next query stay along the decision boundary. At the exploration stage (right), $\boldsymbol{c}$ is the centroid of the initial positive samples, which keeps the queries centered around the existing (real-world) samples rather than biasing towards some direction.

can be identified as quickly as possible. Therefore, to enable continuous exploitation of the decision boundary, we propose the following query strategy

$$\begin{aligned} &\min_{\boldsymbol{x}\in X_U} \; V(\boldsymbol{x}) \\ &\text{s.t. } p_\epsilon(\boldsymbol{x}) \geq \tau \end{aligned} \tag{6.6}$$

where $V(\boldsymbol{x})$ is the predictive variance at $\boldsymbol{x}$, and τ is a threshold of the informativeness measure $p_\epsilon(\boldsymbol{x})$.

Theorem 6.1. *The solution to Eq. 6.6 will lie at the intersection of the estimated decision boundary ($\bar{f}(\boldsymbol{x}) = 0$) and the isocontour of $p_\epsilon(\boldsymbol{x}) = \tau$ (Point A in Fig. 6.2), if that intersection A exists.*

Proof. In the following proof, we denote $\bar{f}_P = \bar{f}(\boldsymbol{x}_P)$, and $V_P = V(\boldsymbol{x}_P)$. For a sample $\boldsymbol{x}_A$ at the intersection of $\bar{f}(\boldsymbol{x}) = 0$ and $p_\epsilon(\boldsymbol{x}) = \tau$, we have $\bar{f}_A = 0$ and $p_\epsilon(\boldsymbol{x}_A) = \Phi(-\epsilon/\sqrt{V_A}) = \tau$ (Point A in Fig. 6.2); and for a sample $\boldsymbol{x}_B$ that is any feasible solution to Eq. 6.6, we have $p_\epsilon(\boldsymbol{x}_B) = \Phi(-(|\bar{f}_B| + \epsilon)/\sqrt{V_B}) \geq \tau$ (Point B in

Fig. 6.2). Thus we get $\epsilon/\sqrt{V_B} \leq (|\bar{f}_B| + \epsilon)/\sqrt{V_B} \leq \epsilon/\sqrt{V_A}$. Therefore, $V_A \leq V_B$. The equality holds when $|\bar{f}_B| = 0$ and $p_\epsilon(\boldsymbol{x}_B) = \tau$, *i.e.*, $\boldsymbol{x}_B$ is also at the intersection of $\bar{f}(\boldsymbol{x}) = 0$ and $p_\epsilon(\boldsymbol{x}) = \tau$. Thus we proved the intersection has the minimal predictive variance among feasible solutions to Eq. 6.6, and hence it is the optimal solution. □

Theorem 6.1 indicates that when applying the query strategy shown in Eq. 6.6, the active learner will only query points at the estimated decision boundary[3] as long as the estimated decision boundary and the isocontour of $p_\epsilon(\boldsymbol{x}) = \tau$ intersect. The fact that this intersection exists indicates that there are points on the decision boundary that are informative to some extent (*i.e.*, with $p_\epsilon(\boldsymbol{x}) \geq \tau$). We call this stage the *exploitation stage* — at this stage the active learner exploits the decision boundary. Equation 6.6 ensures that the queries are always on the estimated decision boundary until the exploitation stage ends (*i.e.*, there are no longer informative points on the decision boundary).

If the estimated decision boundary and the isocontour of $p_\epsilon(\boldsymbol{x}) = \tau$ do not intersect, then the algorithm has fully exploited any informative points on the estimated decision boundary (*i.e.*, for all the points on the estimated decision boundary, we have $p_\epsilon(\boldsymbol{x}) < \tau$). We call this stage the *exploration stage* since the active learner starts to search for another decision boundary (Fig. 6.2). In this stage, we want the new query to be both informative and close to where we started, since we do not want the new query to deviate too far from where we start. Therefore, the query strategy at the exploration stage is

$$\begin{aligned} \min_{\boldsymbol{x} \in X_U} \quad & \|\boldsymbol{x} - \boldsymbol{c}\| \\ \text{s.t. } & p_\epsilon(\boldsymbol{x}) \geq \tau \end{aligned} \tag{6.7}$$

[3]In Sect. 6.3, we assume that the queried point is the exact solution to the query strategy. However since we approximate the exact solution by using a pool-based sampling setting, the query may deviate from the exact solution slightly.

where the objective function is the Euclidean distance between $\boldsymbol{x}$ and a center $\boldsymbol{c}$. This objective keeps the new query selected by Eq. 6.7 close to $\boldsymbol{c}$. In practice, initially when there are only samples from one class, we set $\boldsymbol{c}$ as the initial point $\boldsymbol{x}^{(0)}$ to keep new queries close to where we start; once there are both positive and negative samples, we set $\boldsymbol{c}$ as the centroid of these initial positive samples, since we want to keep new queries close to the initial feasible region.

Theorem 6.2. *Given $\boldsymbol{x}^*$ as the solution to Eq. 6.7, we have $p_\epsilon(\boldsymbol{x}^*) = \tau$, if $p_\epsilon(\boldsymbol{c}) < \tau$.*

Proof. Since $p_\epsilon(\boldsymbol{c}) < \tau$, $\boldsymbol{c}$ itself is not the solution of Eq. 6.9. Thus $\|\boldsymbol{x}^* - \boldsymbol{c}\| > 0$. Then we have $p_\epsilon(\boldsymbol{x}) < \tau$ at any point within a $(d-1)$-sphere centered at $\boldsymbol{c}$ with radius $\|\boldsymbol{x}^* - \boldsymbol{c}\|$, because otherwise the query will be inside the sphere. Thus on that sphere we have $p_\epsilon(\boldsymbol{x}) \leq \tau$. So $p_\epsilon(\boldsymbol{x}^*) \leq \tau$, since $\boldsymbol{x}^*$ is on that sphere. Because $\boldsymbol{x}^*$ is a feasible solution to Eq. 6.9, we also have $p_\epsilon(\boldsymbol{x}^*) \geq \tau$ at $\boldsymbol{x}^*$. Therefore $p_\epsilon(\boldsymbol{x}^*) = \tau$. □

Theorem 6.2 shows that in each iteration, the optimal query $\boldsymbol{x}^*$ selected by Eq. 6.7 is on the isocontour of $p_\epsilon(\boldsymbol{x}) = \tau$.

For both Eq. 6.6 and 6.7, the feasible solutions are in the region of $p_\epsilon(\boldsymbol{x}) \geq \tau$. Intuitively this means that we only query samples with at least some level of informativeness. We call the region where $p_\epsilon(\boldsymbol{x}) \geq \tau$ the *unexplored region*, since it contains informative samples (feasible solutions) that our query strategy cares about; while we call the rest of the input space ($p_\epsilon(\boldsymbol{x}) \leq \tau$) the *explored region* (Fig. 6.2).

The upper bound of $p_\epsilon(\boldsymbol{x})$ is $\Phi(-\epsilon/\sup_{\boldsymbol{x}} V(\boldsymbol{x}))$, and it lies infinitely far away from the labeled samples. In Eq. 6.3, $K + W^{-1}$ is positive semidefinite, thus $\boldsymbol{k}(\boldsymbol{x})^T(K + W^{-1})^{-1}\boldsymbol{k}(\boldsymbol{x}) \geq 0$ and $V(\boldsymbol{x}) \leq k(\boldsymbol{x}, \boldsymbol{x})$. For a kernel $k(\cdot)$ with $k(\boldsymbol{x}, \boldsymbol{x}) = 1$ (*e.g.*, the Gaussian or the exponential kernel), we have $V(\boldsymbol{x}) \leq 1$. Thus $p_\epsilon(\boldsymbol{x}) \leq \Phi(-\epsilon)$. To ensure that Eq. 6.9 has a feasible solution, we have to set $\tau \leq \Phi(-\epsilon)$. Therefore, we

can set $\tau = \Phi(-\eta\epsilon)$, where $\eta \geq 1.0$. Then the constraint in Eq. 6.6 and 6.7 can be expressed as

$$\Phi\left(-\frac{|\bar{f}(\boldsymbol{x})| + \epsilon}{\sqrt{V(\boldsymbol{x})}}\right) \geq \Phi(-\eta\epsilon)$$

which can be written as

$$\eta\epsilon\sqrt{V(\boldsymbol{x})} - |\bar{f}(\boldsymbol{x})| \geq \epsilon \tag{6.8}$$

The left-hand side of Eq. 6.8 is identical to the acquisition function of the *straddle heuristic* when $\eta\epsilon = 1.96$ [26]. The straddle heuristic queries the sample with the largest value of the acquisition function. This acquisition function accounts for the *ambiguity* of samples in terms of their confidence intervals [86]:

$$\begin{aligned} a(\boldsymbol{x}) &= \min\{-\min Q(\boldsymbol{x}), \max Q(\boldsymbol{x})\} \\ &= 1.96\sqrt{V(\boldsymbol{x})} - |\bar{f}(\boldsymbol{x})| \end{aligned}$$

where $Q(\boldsymbol{x})$ is the 95% confidence interval of $\boldsymbol{x}$.

Substituting Eq. 6.8 for the constraint in Eq. 6.6 and 6.7, and combining the exploitation and exploration stages, our overall query strategy becomes

$$\begin{aligned} \min_{\boldsymbol{x}\in X_U} \; & V(\boldsymbol{x})^{\alpha}\|\boldsymbol{x} - \boldsymbol{c}\|^{1-\alpha} \\ \text{s.t. } & \eta\epsilon\sqrt{V(\boldsymbol{x})} - |\bar{f}(\boldsymbol{x})| \geq \epsilon \end{aligned} \tag{6.9}$$

where the indicator α is 1 at the exploitation stage, and 0 otherwise. Section 6.4.4 introduces how to set α (*i.e.*, when to exploit vs explore).

In general, the unbounded domain estimation problem can be solved using a family

of query strategies with the following form

$$\min_{\boldsymbol{x} \in X_U} \; D(\boldsymbol{x}, X_L)$$
$$\text{s.t. } I(\boldsymbol{x}) \geq \tau$$

where $D(\boldsymbol{x}, X_L)$ is a function that increases as $\boldsymbol{x}$ moves away from the labeled samples, and $I(\boldsymbol{x})$ is the informativeness measure that is used in any bounded active learning methods. Our query strategies of Eqn 6.6 and 6.7 all have this form. Comparatively, for bounded active learning methods, the query strategies are usually in the form of $\max_{\boldsymbol{x} \in X_U} I(\boldsymbol{x})$.

6.4 Dynamic Local Pool Generation

We cast our problem as pool-based sampling by generating a pool of unlabeled instances de novo in each iteration. A naïve way to generate this pool is to try to sample points anywhere near the $p_\epsilon(\boldsymbol{x}) = \tau$ isocontour. However, intuitively, as the algorithm searches progressively larger volumes of the input space, the pool volume will likewise expand. This expansion means that the size of the pool will increase dramatically over time (assuming we want a constant sample density). This increase, however, makes the computation of Eq. 6.2 and 6.3 expensive during later expansion stages.

To bypass this problem, we propose a *dynamic local pool* method that generates the pool of candidate samples only at a certain location in each iteration, rather than sampling the entire domain.[4] The key insight behind our local pooling method is that while the optimal solution to Eq. 6.9 can, in principle, occur anywhere on

[4]Sampling methods like random sampling or Poisson-disc sampling [23] can be used to generate the pool. We use random sampling here thereby for simplicity. The specific choice of the sampling method within the local pool is not central to the overall method.

the $p_\epsilon(\boldsymbol{x}) = \tau$ isocontour, in practice, multiple points on the isocontour are equally optimal. All we need to do is sample points around any one of those optima. Below, we derive guarantees for how to sample volumes near one of those optima, thus only needing to sample a small fraction of the total domain volume.

6.4.1 Scope of an Optimal Query

Theorem 6.3. *Let δ be the distance between an optimal query[5] and its nearest labeled sample. We have*

$$\delta < \beta l \tag{6.10}$$

where β is a coefficient depends on ϵ, η, and the GP model.

[5]The optimal query means the exact solution to the AES query strategy shown in Eq. 6.6, 6.7, or 6.9

Proof. According to Eq. 6.2, given an optimal query $\boldsymbol{x}^*$, we have

$$
\begin{aligned}
|\bar{f}(\boldsymbol{x}^*)| &= |\boldsymbol{k}(\boldsymbol{x}^*)^T \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}})| \\
&= \left| \boldsymbol{k}(\boldsymbol{x}^*)^T \nabla \log \begin{bmatrix} \Phi(y_1 f_1) \\ \vdots \\ \Phi(y_{t-1} f_{t-1}) \end{bmatrix} \right| \\
&= \left| \boldsymbol{k}(\boldsymbol{x}^*)^T \begin{bmatrix} y_1 \mathcal{N}(f_1)/\Phi(y_1 f_1) \\ \vdots \\ y_{t-1} \mathcal{N}(f_{t-1})/\Phi(y_{t-1} f_{t-1}) \end{bmatrix} \right| \\
&= \left| \sum_{i=1}^{t-1} k(\boldsymbol{x}^*, \boldsymbol{x}^{(i)}) y_i \frac{\mathcal{N}(f_i)}{\Phi(y_i f_i)} \right| \\
&\leq \sum_{i=1}^{t-1} \left| k(\boldsymbol{x}^*, \boldsymbol{x}^{(i)}) y_i \frac{\mathcal{N}(f_i)}{\Phi(y_i f_i)} \right| \\
&< \sum_{i=1}^{t-1} k_m \text{sign}(y_i) y_i \frac{\mathcal{N}(f_i)}{\Phi(y_i f_i)} \\
&= k_m \text{sign}(\boldsymbol{y})^T \begin{bmatrix} y_1 \mathcal{N}(f_1)/\Phi(y_1 f_1) \\ \vdots \\ y_{t-1} \mathcal{N}(f_{t-1})/\Phi(y_{t-1} f_{t-1}) \end{bmatrix} \\
&= k_m \mu
\end{aligned}
$$

where

$$
\begin{aligned}
k_m &= \max_{\boldsymbol{x}^{(i)} \in X_L} k(\boldsymbol{x}^*, \boldsymbol{x}^{(i)}) \\
&= \exp\left(-\frac{\min_{\boldsymbol{x}^{(i)} \in X_L} \|\boldsymbol{x}^* - \boldsymbol{x}^{(i)}\|^2}{2l^2} \right) \\
&= e^{-\delta^2/(2l^2)}
\end{aligned} \tag{6.11}
$$

and

$$
\mu = \text{sign}(\boldsymbol{y})^T \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{6.12}
$$

Similarly,

$$\begin{aligned} V(\boldsymbol{x}^*) &= 1 - \boldsymbol{k}(\boldsymbol{x}^*)^T (K + W^{-1})^{-1} \boldsymbol{k}(\boldsymbol{x}^*) \\ &> 1 - (k_m \mathbf{1})^T (K + W^{-1})^{-1} (k_m \mathbf{1}) \\ &= 1 - k_m^2 \mathbf{1}^T (K + W^{-1})^{-1} \mathbf{1} \\ &= 1 - k_m^2 \nu \end{aligned} \tag{6.13}$$

where

$$\nu = \mathbf{1}^T (K + W^{-1})^{-1} \mathbf{1} \tag{6.14}$$

Therefore for the optimal query $\boldsymbol{x}^*$ we have

$$p_\epsilon(\boldsymbol{x}^*) = \Phi\left(-\frac{|\bar{f}(\boldsymbol{x}^*)| + \epsilon}{\sqrt{V(\boldsymbol{x}^*)}} \right) > \Phi\left(-\frac{k_m \mu + \epsilon}{\sqrt{1 - k_m^2 \nu}} \right)$$

Both Theorem 6.1 and 6.2 state that $p_\epsilon(\boldsymbol{x}^*) = \tau$, thus

$$\Phi\left(-\frac{k_m \mu + \epsilon}{\sqrt{1 - k_m^2 \nu}} \right) < \tau$$

When $\tau = \Phi(-\eta\epsilon)$, we have

$$\frac{k_m \mu + \epsilon}{\sqrt{1 - k_m^2 \nu}} > \eta\epsilon \tag{6.15}$$

Plugging Eq. 6.11 into Eq. 6.15 and solving for the distance δ, we get

$$\delta < \beta l$$

where

$$\beta = \sqrt{2 \log \frac{\mu^2 + \eta^2 \epsilon^2 \nu}{\eta\epsilon \sqrt{\mu^2 + (\eta^2 - 1)\epsilon^2 \nu} - \epsilon\mu}} \tag{6.16}$$

□

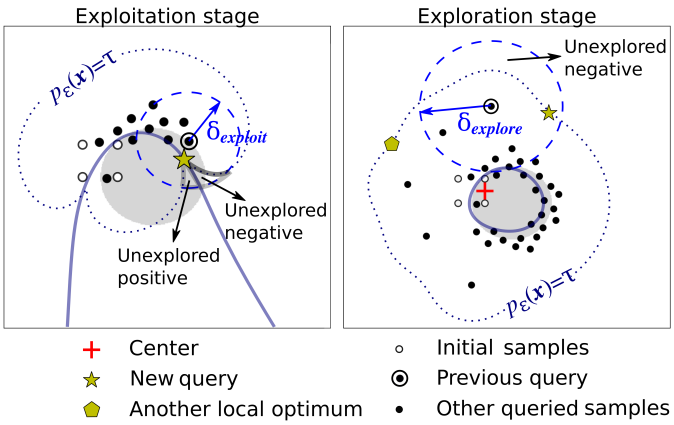


Figure 6.3: Dynamic local pools (dashed circles) at the exploitation stage (left) and the exploration stage (right). During the exploitation stage, the estimated decision boundary divided the unexplored region into two subregions: unexplored negative $\mathcal{R}_1 = \{\boldsymbol{x}|p_\epsilon(\boldsymbol{x}) > \tau, \bar{f}(\boldsymbol{x}) < 0\}$ and unexplored positive $\mathcal{R}_2 = \{\boldsymbol{x}|p_\epsilon(\boldsymbol{x}) > \tau, \bar{f}(\boldsymbol{x}) > 0\}$; while during the exploration stage, there will be at most one of $\mathcal{R}_1$ and $\mathcal{R}_2$ in the unexplored region. This property can be used to distinguish between the exploitation/exploration stages.

Theorem 6.3 indicates that if we set the pool boundary by extending the current labeled sample range by βl, then that pool is guaranteed to contain all solutions to Eq. 6.9; that is, extending the overall pool boundary further will not increase the chances of sampling near $p_\epsilon(\boldsymbol{x}) = \tau$, and will only decrease the sample density (given a fixed pool size) or increase the evaluated samples (given a fixed sample density). However, if we generate the pool based solely on this principle (*i.e.*, extending the current labeled sample range by βl), the pool size will still increase over time as the domain size grows. The next two sections show how, for the exploration and exploitation stages respectively, we can further reduce the sample boundary to only a local hyper-sphere.

6.4.2 Pool for the Exploration Stage

Theorem 6.4. *During the exploration stage of Active Expansion Sampling, the distance between an optimal query and its nearest labeled sample is*

$$\delta < \delta_{explore} = \beta l \tag{6.17}$$

Theorem 6.4 is derived from Eq. 6.10. The nearest labeled sample of the optimal query could be any border point (a sample lying on the periphery of the labeled set). There are multiple local optima that are equally useful for expanding the explored region (Fig. 6.3). Thus we just sample near one of those optima. Specifically, we approximate the nearest labeled sample as the previous query. With this approximation, incorporating Theorem 6.4, the optimal query will be inside $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{explore})$, the $(d-1)$-sphere with a radius of $\delta_{explore}$, centered at the previous query $\boldsymbol{x}^{(t-1)}$. Thus during the exploration stage, we set the pool boundary to be that sphere (Fig. 6.3).

Sometimes when AES switches from exploitation to exploration, the previous query may not lie on the periphery of the labeled samples. This causes samples around the previous query to have low values of $p_\epsilon(\boldsymbol{x})$. In this case, there might not be a feasible solution to Eq. 6.9. Thus, every time AES switches from exploitation to exploration, we center the pool around the farthest labeled sample from the centroid of the initial positive samples (*i.e.*, $\text{argmax}_{\boldsymbol{x} \in X_L} \|\boldsymbol{x} - \boldsymbol{c}\|$). This ensures that AES generates pool samples near the periphery of the labeled samples.

6.4.3 Pool for the Exploitation Stage

Theorem 6.5. *During the exploitation stage of Active Expansion Sampling, the distance between an optimal query and its nearest labeled sample is*

$$\delta < \delta_{exploit} = \gamma l \tag{6.18}$$

where γ is a coefficient depends on ϵ, η, and the GP model.

Proof. Theorem 6.1 states that the optimal query in the exploitation stage lies at the intersection of $\bar{f}(\boldsymbol{x}) = 0$ and $p_\epsilon(\boldsymbol{x}) = \tau$. By substituting $\Phi(-\eta\epsilon)$ for τ, we have

$$V(\boldsymbol{x}^*) = \frac{1}{\eta^2} \tag{6.19}$$

According to Eq. 6.13, we have $V(\boldsymbol{x}^*) > 1 - k_m^2\nu$. Combining Eq. 6.11, 6.14, and 6.19, we get

$$\delta < \delta_{exploit} = \gamma l$$

where

$$\gamma = \sqrt{\log \frac{\eta^2\nu}{\eta^2 - 1}} \tag{6.20}$$

□

Similar to the exploration stage, based on Theorem 6.5, we define the pool boundary during the exploitation stage as $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{exploit})$, a $(d-1)$-sphere with a radius of $\delta_{exploit}$, centered at the previous query $\boldsymbol{x}^{(t-1)}$ (Fig. 6.3).

6.4.4 Choosing when to Exploit versus Explore

Since we use different rules to generate the pool at the exploitation and exploration stage, we need to distinguish between the two stages at the beginning of each iteration. In the exploitation stage, according to Theorem 6.5, the optimal query lies within the $(d-1)$-sphere $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{exploit})$ centered at the previous query. While, according to Theorem 6.1, that same query must lie where the estimated decision boundary and the isocontour of $p_\epsilon(\boldsymbol{x}) = \tau$ intersect. Thus, the decision boundary and the isocontour divide the sphere $\mathcal{C}$ into four regions (Fig. 6.3):

unexplored negative $\mathcal{R}_1 = \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) > \tau, \bar{f}(\boldsymbol{x}) < 0\}$;

unexplored positive $\mathcal{R}_2 = \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) > \tau, \bar{f}(\boldsymbol{x}) > 0\}$;

explored negative $\mathcal{R}_3 = \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) < \tau, \bar{f}(\boldsymbol{x}) < 0\}$; and

explored positive $\mathcal{R}_4 = \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) < \tau, \bar{f}(\boldsymbol{x}) > 0\}$.

In contrast, during exploration the estimated decision boundary and the $p_\epsilon(\boldsymbol{x}) = \tau$ isocontour do not intersect — meaning, unlike exploitation, there exist only two of the four regions (either $\mathcal{R}_1$ & $\mathcal{R}_3$ *or* $\mathcal{R}_2$ & $\mathcal{R}_4$). In particular, within the unexplored region, $\bar{f}(\boldsymbol{x})$ will be either all positive or all negative, *i.e.*, $\mathcal{R}_1$ and $\mathcal{R}_2$ cannot exist simultaneously (Fig. 6.3).

We use this property to detect exploitation or exploration by generating a pool (a set of uniformly distributed samples) within the boundary $\mathcal{C}(\boldsymbol{x}^{(t-1)}, \delta_{exploit})$ and checking if, for samples with $p_\epsilon(\boldsymbol{x}) > \tau$, samples differ in $\bar{f}(\boldsymbol{x}) > 0$ and $\bar{f}(\boldsymbol{x}) < 0$. If so, AES is in the exploitation stage; otherwise it is in the exploration stage.

6.5 Theoretical Analysis

In this section, we derive a theoretical accuracy bound for AES with respect to its hyperparameters. We further discuss the influence of those hyperparameters on the classification accuracy, the query density, and the exploration speed. The results of this section guide the selection of proper hyperparameters given an accuracy or budget requirement.

6.5.1 Accuracy Analysis

It is impossible to discuss the function accuracy across the entire input space, since the input space is unbounded. However, we can consider ways to bound the accuracy within bounded explored regions at any time step.

As mentioned in Sect. 6.3.1, $p_\epsilon(\boldsymbol{x}) = P(L(\boldsymbol{x}) > \epsilon)$, where $L(\boldsymbol{x})$ is the misclassification loss at $\boldsymbol{x}$ defined in Eq. 6.5. Thus within the explored region, we have

$$P(L(\boldsymbol{x}) \geq \epsilon) \leq \tau \quad \forall \boldsymbol{x} \in \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) \leq \tau\}$$

or

$$P(L(\boldsymbol{x}) \leq \epsilon) \geq 1 - \tau \quad \forall \boldsymbol{x} \in \{\boldsymbol{x} | p_\epsilon(\boldsymbol{x}) \leq \tau\} \tag{6.21}$$

This shows that at any location within the explored region of the input space, the proposed method guarantees an upper bound ϵ of misclassification loss with a probability of at least $1 - \tau$ at any given point. Since, in the exploration stage, the estimated decision boundary lies inside the $p_\epsilon(\boldsymbol{x}) \leq \tau$ region (as discussed in Sect. 6.3.2), we have

$$P(L(\boldsymbol{x}) \leq \epsilon) \geq 1 - \tau \quad \forall \boldsymbol{x} \in \{\boldsymbol{x} | \bar{f}(\boldsymbol{x}) = 0\}$$

This means that in the exploration stage, the estimated decision boundary $\bar{f}(\boldsymbol{x}) = 0$ lies in between the isocontours of $f(\boldsymbol{x}) = \pm\epsilon$ with a probability of at least $1 - \tau$, where f is the true latent function.

Note that Eq. 6.21 shows that AES's accuracy bound within the explored region is independent of the number of iterations or labeled samples. One advantage of keeping a constant accuracy bound for AES is that the accuracy in the explored region meets our requirements[6] whenever AES stops. This also means that the estimation within the explored region is reliable at any iteration (although this is not true if one includes the unexplored region). In contrast, bounded active learning methods usually only achieve required accuracy after a certain number of iterations, before which the estimation may not be reliable. Therefore, AES can be used for real-time prediction of samples' feasibility in the explored region.

6.5.2 Query Density

In Gaussian Processes, given a fixed homoscedastic Gaussian or exponential kernel, we can measure the query density by looking at the predictive variance at queried points. According to Eq. 6.3, $V(\boldsymbol{x})$ only depends on $\boldsymbol{k}(\boldsymbol{x})$, which is affected by the distances between $\boldsymbol{x}$ and other queries. A smaller variance at a query indicates that it is closer to other queries, and hence a higher query density; and vise versa.

Theorem 6.6. *The predictive variance of an optimal query in the exploitation and exploration stage is*

$$V(\boldsymbol{x}_{exploit}) = \frac{1}{\eta^2} \tag{6.22}$$

[6]We can set ϵ and τ such that the accuracy bound is as required. Details about how to set hyperparameters are in Sect. 6.5.3.

and

$$V(\boldsymbol{x}_{explore}) = \frac{1}{\eta^2}\left(1 + \frac{|\bar{f}(\boldsymbol{x}_{explore})|}{\epsilon}\right)^2 \tag{6.23}$$

where $\boldsymbol{x}_{exploit}$ and $\boldsymbol{x}_{explore}$ are optimal queries at the exploitation stage and exploration stage, respectively.

Proof. According to Eq. 6.19, the predictive variance of an optimal query $\boldsymbol{x}_{exploit}$ in the exploitation stage is

$$V(\boldsymbol{x}_{exploit}) = \frac{1}{\eta^2}$$

While in the exploration stage, we have $p_\epsilon(\boldsymbol{x}_{explore}) = \tau$ at the optimal query $\boldsymbol{x}_{explore}$ (Theorem 6.2). And by applying Eq. 6.4 and setting $\tau = \Phi(-\eta\epsilon)$, we have

$$V(\boldsymbol{x}_{explore}) = \frac{1}{\eta^2}\left(1 + \frac{|\bar{f}(\boldsymbol{x}_{explore})|}{\epsilon}\right)^2$$

□

This theorem indicates that the predictive variances of queries at the exploitation stage are always smaller than those at the exploration stage (as $|\bar{f}(\boldsymbol{x}_{explore})| > 0$). Thus the query density at the exploitation stage is always higher than that at the exploration stage. The property of having a denser set of points along the decision boundary (queried during the exploitation stage) and a sparser set of points at other regions (queried during the exploration stage) is desirable because we want to save our query budget for refining the decision boundary rather than other regions of the input space.

Equation 6.21 and 6.22 also reflect the trade-off between the accuracy and the running time. When the query density near the decision boundary is high (small $V(\boldsymbol{x}_{exploit})$ in Eq. 6.22), η is large, thus τ in Eq. 6.21 is small, which means our model will have a higher probability of having a misclassification loss less than ϵ. However,

as the query density gets higher, we need more queries to cover a certain region, thus the running time increases.

6.5.3 Influence of Hyperparameters

There are four hyperparameters that control Active Expansion Sampling — the initial point $\boldsymbol{x}^{(0)}$, ϵ and η in the exploitation/exploration stage, and the length scale l of the GP kernel. The choice of the kernel function and length scale depends on assumptions regarding the nature and smoothness of the underlying feasibility function. Such kernel choices have been covered extensively in prior research and we refer interested readers to [172] for multiple methods of choosing l. Note that it is difficult to optimize the length scale at each iteration, since the length scale will eventually be pushed to extremes. In the exploitation stage, for example, once the length scale is smaller than the previous iteration, the distance between the new query and its nearest query will also be smaller (due to Eq. 12). Then the maximum marginal likelihood estimation will result in a smaller length scale, as the estimated function is steeper. This process will repeat and eventually cause the optimal length scale to converge to 0. The initial point $\boldsymbol{x}^{(0)}$ can be any point not too far away from the boundary of feasible regions, since otherwise it will take a large budget to just search for a sample from the opposite class. Here we focus on the analysis of the other two hyperparameters — ϵ and η.

According to Eq. 6.21, ϵ and τ affect the classification accuracy in a probabilistic way. When $\tau = \Phi(-\eta\epsilon)$, we have $P(L(\boldsymbol{x}) \leq \epsilon) \geq 1 - \Phi(-\eta\epsilon)$ in the explored region. This offers us a guideline for setting ϵ and η with respect to a given accuracy requirement.

According to Eq. 6.22 and 6.23, η controls the density of queries in both ex-

ploitation and exploration stages. Specifically, as we increase η, $V_{exploit}$ and $V_{explore}$ decreases, increasing the query density and essentially placing labeled points closer together.

In contrast, ϵ only controls the distances between queries in the exploration stage[7]. Increasing ϵ decreases $V_{explore}$ and hence increases the density of queries in the exploration stage. This density of queries affects (1) how fast we can expand the explored region, and (2) how likely we are to capture small feasible regions. When η or ϵ increases, we expand the explored region slower, making it more likely that we will discover smaller feasible regions. Likewise, we also slow down the expansion in exploitation stages, making the classifier more likely to capture a sudden change along domain boundaries.

Note that when $\epsilon = 0$, the constraint of $p_\epsilon(\boldsymbol{x}) \geq \tau$ in Eq. 6.9 is equivalent to $\bar{f}(\boldsymbol{x}) = 0$, thus theoretically all queries should lie near the estimated decision boundary. In this case, the Active Expansion Sampling acts like *Uncertainty Sampling* [135, 136]. In practice, however, AES will be unable to find a feasible solution when $\epsilon = 0$ since no candidate sample will be exactly on the decision boundary under the pool-based sampling setting.

6.6 Using Design Manifolds to Synthesize Novel Designs

It is difficult to directly explore a real-world design space since it is usually high-dimensional, and most points in that space will be unrealistic or nonfunctional. To handle real-world design spaces, our proposed method assumes that design variables original expressed in a high-dimensional design space $\mathcal{X}$ usually lie on a lower-

[7]Technically, due to sampling error introduced when generating the pool, the exploitation stage will be influenced by ϵ (since $\bar{f}(\boldsymbol{x}^*)$ is only ≈ 0). But this effect is negligible compared to ϵ's influence on the exploration stage.

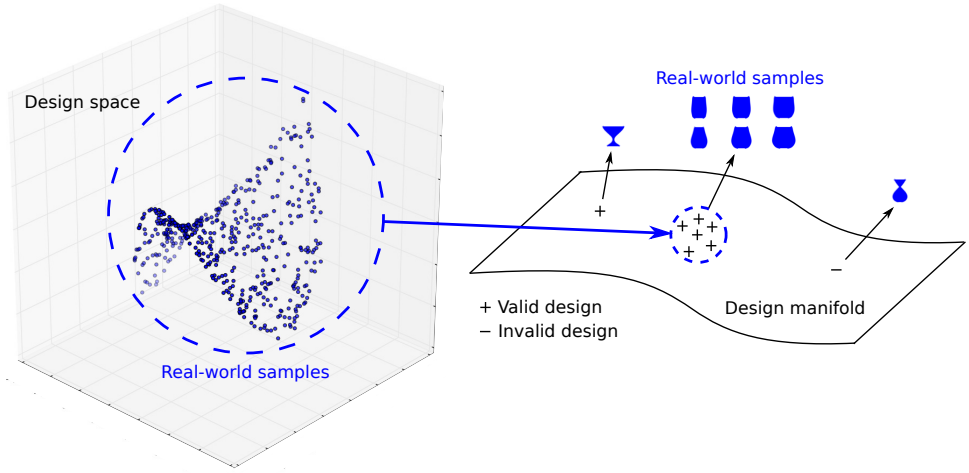


Figure 6.4: 3D visualization of high dimensional design space showing that design variables actually lie on a 2-dimensional manifold [41, 49]. At a point away from the real-world stemless glass samples, the glass contours are self-intersecting; at another point, the shape becomes a stem glass.

dimensional *design manifold* (Fig. 6.4) [17, 41, 49, 216]. We can thus project designs on that lower-dimensional manifold $\mathcal{F}$ by a mapping $g : \mathcal{X} \to \mathcal{F}$. Then given any point in that space, we can synthesize new designs by a reverse mapping $g' : \mathcal{F} \to \mathcal{X}$. This reduces the problem of exploring the high-dimensional design space $\mathcal{X}$ to exploring a corresponding embedding space $\mathcal{F}$.

Just like in the original design space, there are boundaries for feasible designs in the embedded space. The function that evaluates feasible domains can thus be expressed as $h : \mathcal{F} \to \{-1, 1\}$. Figure 6.4 shows an example of a glassware design manifold, where the synthesized contours are self-intersecting at a point away from the real-world samples. We call these *invalid designs*—designs that are unrealistic or nonfunctional in the real-world. Since real-world samples are all valid designs, normally designs lying between any two real-world samples (*i.e.*, inside the convex hull of all the real-world samples in the embedding space) will also be valid [41, 49]. However, this assumption may not hold for designs that lie *on the manifold* but *beyond* the real-world samples. Since they are on the manifold, they obey similar

rules of variation and deformation learned from the real-world samples, but because they are away from the real-world samples, there is no guarantee that those designs are functional or aesthetically valid.

On a manifold that preserves pairwise distances between samples in the design space, designs far away from the real-world samples will look different from them [41, 49]. If these designs remain valid, despite being far away from the real-world samples, then we are discovering innovative designs. The methods presented in this chapter are one way to achieve this creative exploration, even in high dimensional design spaces. Note that this method assumes that the dimensionality of the design space can be reduced. In cases where this assumption does not hold, we have to directly explore the original high-dimensional design space.

6.7 Experimental Evaluation

We evaluate the performance of AES in capturing feasible domains using both synthesized and real-world examples. The performance is measured by the F1 score, which is expressed as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

and

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

We compare AES with two conventional bounded adaptive sampling methods — the Neighborhood-Voronoi (NV) algorithm [195] and the straddle heuristic [26]. We also investigate the effects of noise and dimensionality on AES.

We use the same pool size (500 candidate samples[8]) in all the experiments. In Fig. 6.8-6.11 and 6.14, the F1 scores are averaged over 100 runs. We run all 2-dimensional experiments on a Dell Precision Tower 5810 with 16 GB RAM, a 3.5 GHz Intel Xeon CPU E5-1620 v3 processor, and a Ubuntu 16.04 operating system. We run all higher-dimensional experiments on a Dell Precision Tower 7810 with 32 GB RAM, a 2.4 GHz Intel Xeon CPU E5-2620 v3 processor, and a Red Hat Enterprise Linux Workstation 7.2 operating system. The Python code needed to reproduce our AES algorithm, our baseline implementations of NV and Straddle, and all of our below experiments is available at https://github.com/IDEALLab/Active-Expansion-Sampling.

6.7.1 Effect of Hyperparameters

We first use two 2-dimensional test functions — the Branin function and Hosaki function, respectively — as indicator functions to evaluate whether an input is inside the feasible domain. Both examples construct an input space with multiple disconnected feasible regions, which makes the feasible domain identification task challenging.

The Branin function is

$$g(\boldsymbol{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

We define the label $y = 1$ if $\boldsymbol{x} \in \{\boldsymbol{x}|g(\boldsymbol{x}) \leq 8, -9 < x_1 < 14, -7 < x_2 < 17\}$; and $y = -1$ otherwise. The resulting feasible domains resemble three isolated feasible regions (Fig. 6.5). The initial point $\boldsymbol{x}^{(0)} = (3, 3)$. For the Gaussian process, we use a

[8]For NV algorithm, its pool size refers to the test samples generated for the Monte Carlo simulation.

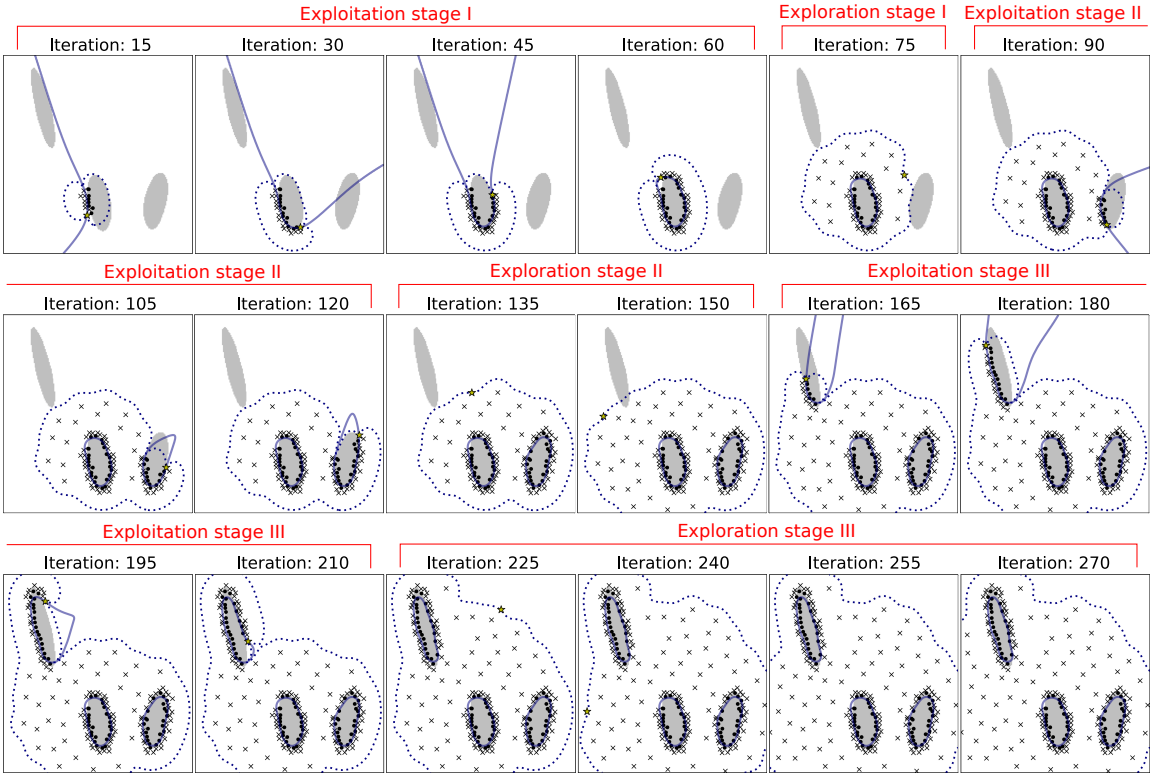
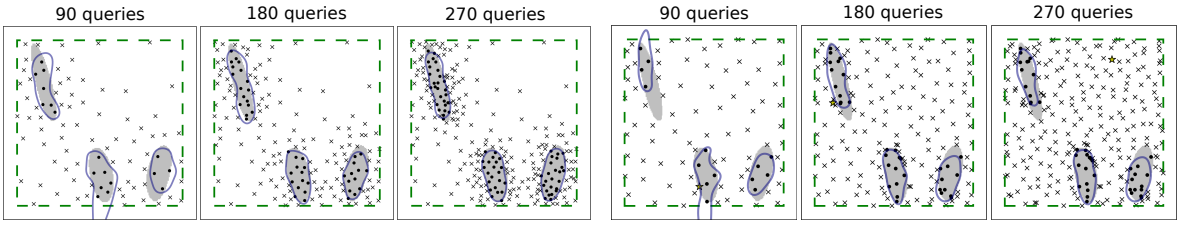


Figure 6.5: Querying sequence for Active Expansion Sampling ($\epsilon = 0.5$ and $\eta = 1.3$). The solid lines are estimated decision boundaries, and the dotted lines are the isocontour $p_\epsilon(\boldsymbol{x}) = \tau$. The gray areas are actual feasible regions.



(a) Neighborhood-Voronoi algorithm. (b) Straddle heuristic.

Figure 6.6: Querying sequence for bounded adaptive sampling methods. The dashed lines are pool boundaries.

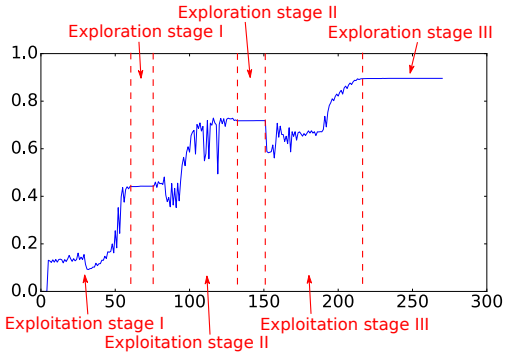
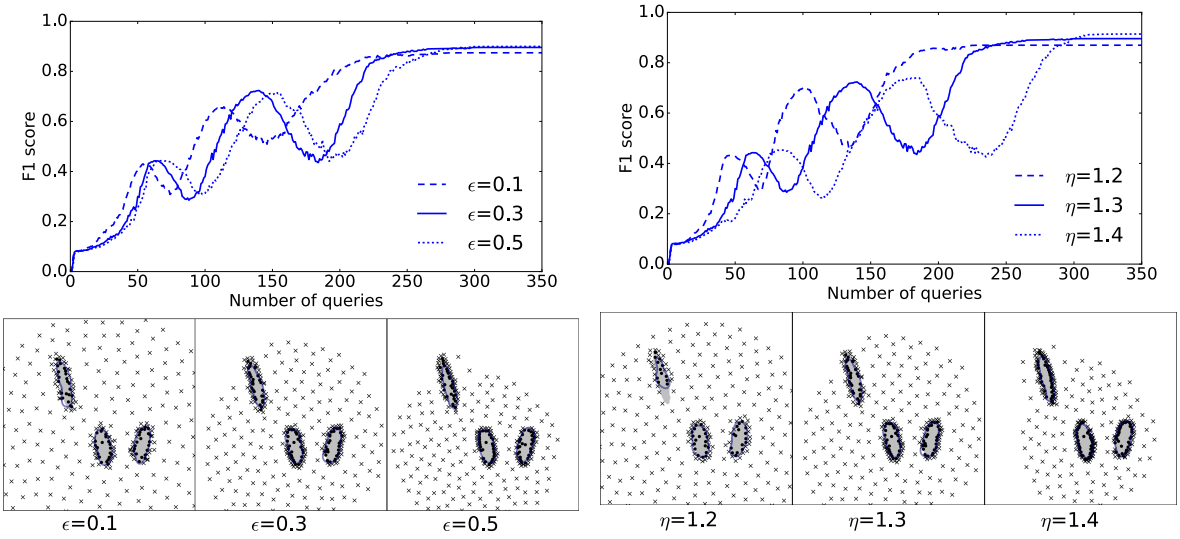


Figure 6.7: F1 score plot for Fig. 6.5. During exploitation stages, the F1 score increases stochastically as the decision boundary changes; while in the exploration stage, the current decision boundaries have been exploited and do not change, thus the F1 score also does not change.



(a) Changing ϵ ($\eta = 1.3$). (b) Changing η ($\epsilon = 0.3$).

Figure 6.8: AES with different ϵ and η on the Branin example. The upper plots show their F1 scores averaged over 100 runs. The lower plots show queried points during one of the 100 runs.

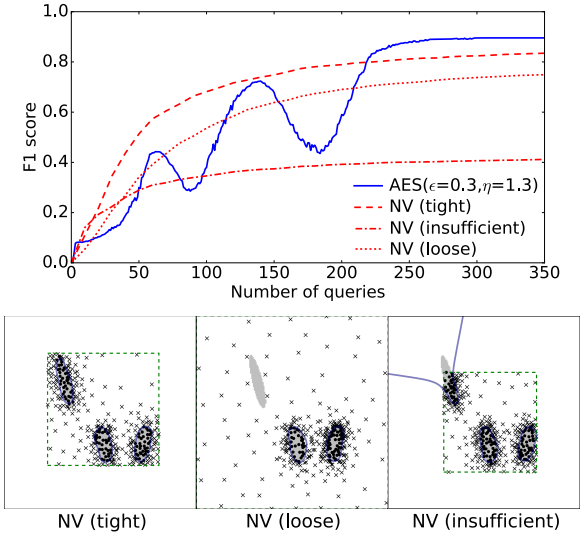


Figure 6.9: AES and NV (with different input variable bounds) on the Branin example. The pool boundaries set in the Neighborhood-Voronoi algorithm are shown as dashed lines.

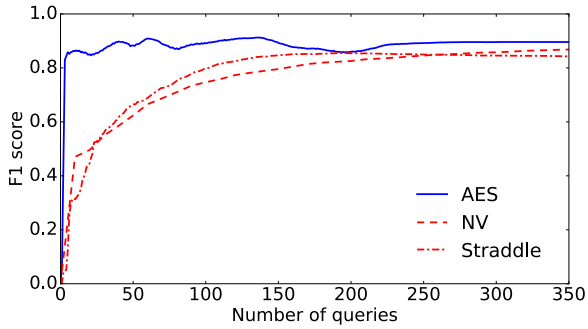
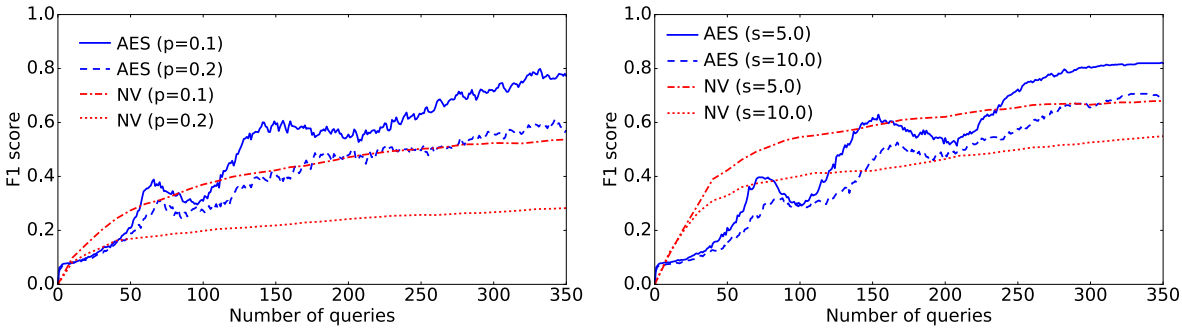


Figure 6.10: F1 scores of AES ($\epsilon = 0.3$ and $\eta = 1.3$), NV, and Straddle (with tight bounds) on the Branin example within the explored region (*i.e.*, the $p_\epsilon(\boldsymbol{x}) < \tau$ region, where $\tau = \Phi(-\eta\epsilon)$, $\epsilon = 0.3$, and $\eta = 1.3$).



(a) Bernoulli noise. (b) Gaussian noise.

Figure 6.11: AES and NV on the Branin example using noisy labels.

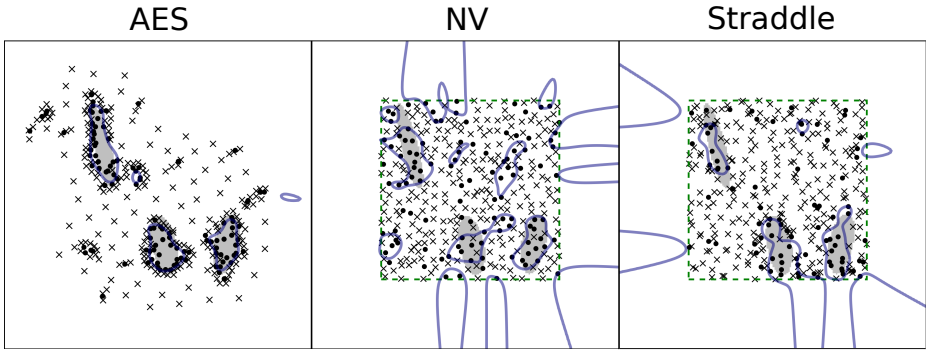


Figure 6.12: Queried points under uniform label noise ($p = 0.2$).

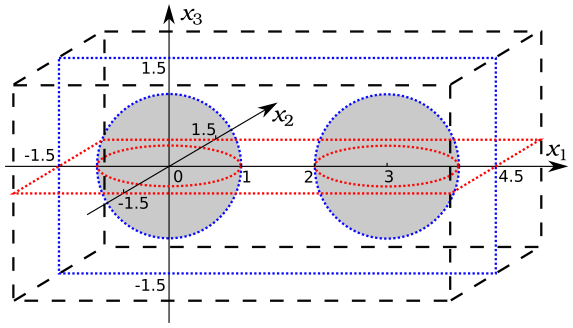
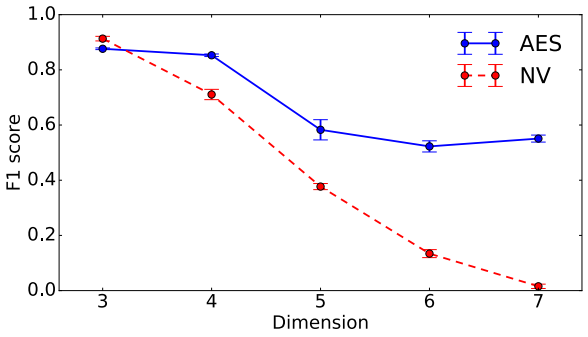
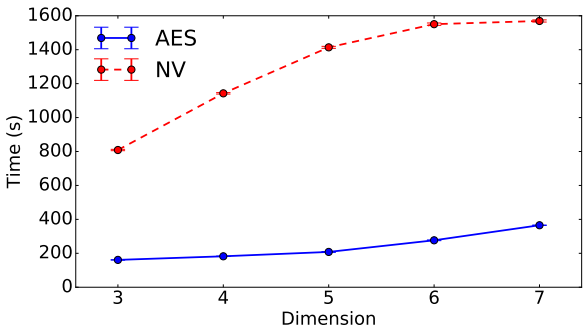


Figure 6.13: The 3-dimensional double-sphere example. The gray regions are the feasible domains. The dashed boxes are the input space bounds for the NV algorithm.



(a) F1 scores.



(b) Total running time.

Figure 6.14: AES and NV on high-dimensional double-sphere examples.

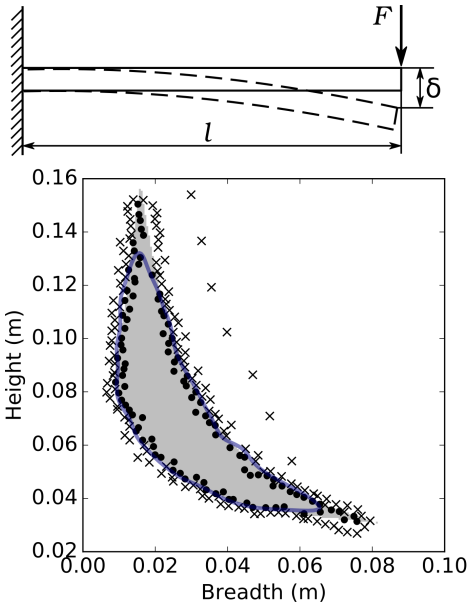


Figure 6.15: AES on the Nowacki beam example.

Gaussian kernel (Eq. 6.1). We set the kernel length scale $l = 0.9$. To compute the F1 scores, we generate samples along a 100×100 grid as the test set in the region where $x_1 \in [-13, 18]$ and $x_2 \in [-8, 23]$.

This section mainly describes the Branin example — as both the Branin and Hosaki examples show similar results — however, we direct interested readers to Sect. 6.8.1, where we describe the Hosaki example in detail and show its experimental results.

For both examples, we use three levels of ϵ (0.1, 0.3, 0.5) and η (1.2, 1.3, 1.4) to demonstrate their effects on AES's performance.

Figures 6.5 and 6.6 show the sequence of queries selected by AES and the two bounded adaptive sampling methods, respectively, applied on the Branin example. For AES, there are three exploitation stages, as there are three disconnected feasible domains. AES starts by querying samples along the initial estimated decision boundary, and then expands queries outward to discover other feasible regions. In contrast, the straddle heuristic simultaneously explores the whole bounded input space, and

Table 6.1: Input space bounds for the NV algorithm and the straddle heuristic (Branin example).

	Tight	Loose	Insufficient
Branin	$x_1 \in [-9, 14]$, $x_2 \in [-7, 17]$	$x_1 \in [-14, 19]$, $x_2 \in [-12, 22]$	$x_1 \in [-4, 9]$, $x_2 \in [-2, 12]$

refines all three decision boundaries. Fig. 6.7 shows the corresponding F1 scores of the experiment in Fig. 6.5. During exploitation stages, AES's F1 score non-monotonically increases as part of the estimated decision boundary is outside the explored region (where AES has confidence in the accuracy); while in the exploration stage, the current decision boundaries are inside the explored region and remain unchanged, thus the F1 score stabilizes.

Figures 6.8a and 6.8b demonstrate the effects of hyperparameters ϵ and η, respectively, on AES's performance. Increasing ϵ or η leads to slower expansion of the explored region and a higher F1 score. This means that using a higher ϵ or η enables accuracy improvement but requires a larger query budget. In both examples, the F1 score is more sensitive to η than ϵ.

6.7.2 Unbounded versus Bounded

We use the NV algorithm and the straddle heuristic as examples of bounded adaptive sampling methods. Because these two methods do not progressively expand the region (as in AES), but rather assumes a fixed region, we create a "bounding box" in the input space, and generate queries inside this box.

When comparing AES with the bounded methods, we use $\epsilon = 0.3$ and $\eta = 1.3$ for AES. In each experiment, we change the size of the input space bounds to evaluate the effect of bound size on these methods. Specifically, we simulate the cases where

we set *tight*, *loose*, and *insufficient* bounds, as shown in Tab. 6.1 and Fig. 6.9. "Tight" means the bounds cover the entire feasible domain while being no larger than needed (in practice we use bounds slightly larger than this to ensure the feasible domain boundary is inside the tight bounds); "loose" means the bounds cover the entire feasible domain but are larger than the tight bounds; "insufficient" means the variable bounds do not cover the entire feasible domain.

As shown in Fig. 6.9, the NV algorithm makes fast accuracy improvement at early stages, and slows down after some iterations. The F1 score of NV is almost monotonically increasing; while AES's score fluctuates because it focuses first on refining the domains it knows about during exploitation (at the expense of accuracy on domains it has not seen yet). This causes AES to have a lower F1 score early on. For the NV algorithm, when the input variable bounds are set properly, both AES and NV achieve similar final F1 scores. However, NV requires more iterations to achieve similar final accuracy to AES, especially when the bounds are set too large, where NV exhausts its query budget exploring unknown regions. When the bounds are set too small to cover certain feasible regions, NV stops improving the F1 score when it begins to over-sample the space and is unable to reach similar accuracy as AES. Note that in this case, we purposefully set the bounds such that they cover the vast majority of the feasible region, leaving only a small feasible area outside of those bounds. Our explicit purpose here is to demonstrate how sensitive such bounded heuristics can be when their bounds are misspecified (even by small amounts). The performance of bounded methods degrades rapidly as their bound sizes decrease further.

Although AES shows slow accuracy improvement over the entire test region, it keeps a constant accuracy bound within the explored region (as discussed in Sect. 6.5.1). Fig. 6.10 shows the F1 scores within the $p_\epsilon(\boldsymbol{x}) < \tau$ region, which is AES's explored region. Specifically, we set $\epsilon = 0.3$, $\eta = 1.3$, and $\tau = \Phi(-\eta\epsilon)$. For the

Table 6.2: Final F1 scores and running time (Branin example).

		F1 score	Time (s)
Branin (350 queries)	AES ($\epsilon = 0.3, \eta = 1.3$)	0.90 ± 0.004	92.34 ± 0.62
	AES ($\epsilon = 0.1, \eta = 1.3$)	0.87 ± 0.008	95.71 ± 0.37
	AES ($\epsilon = 0.5, \eta = 1.3$)	0.90 ± 0.002	89.71 ± 0.38
	AES ($\epsilon = 0.3, \eta = 1.2$)	0.87 ± 0.006	96.73 ± 0.26
	AES ($\epsilon = 0.3, \eta = 1.4$)	0.91 ± 0.002	80.70 ± 0.33
	NV (tight)	0.83 ± 0.021	64.40 ± 0.09
	NV (loose)	0.75 ± 0.030	63.68 ± 0.06
	NV (insufficient)	0.41 ± 0.028	63.83 ± 0.06
	Straddle (tight)	0.82 ± 0.012	43.72 ± 0.22
	Straddle (loose)	0.71 ± 0.014	41.72 ± 0.22
	Straddle (insufficient)	0.34 ± 0.009	54.44 ± 0.21

NV algorithm, we use the tight input space bounds from the previous experiments. By just considering the explored region, AES's F1 scores are quite stable throughout the sampling sequence; while NV's F1 scores are low at the beginning, and then increase until stable.[9] Since AES's accuracy inside the explored region is invariant of the number of iterations, it can be used for real-time prediction of samples' feasibility in the explored region.

Table 6.2 shows the final F1 scores and wall-clock running time of AES, NV, and the straddle heuristic. Note that the confidence interval for NV's averaged F1 scores are much larger than AES. This is because during some runs NV fails to discover all the three feasible regions (Fig. 6.9 for example).

6.7.3 Effect of Noise

Label noise is usually inevitable in active learning tasks. The noise comes from, for example, simulation/experimental error or human annotators' mistakes. We test

[9]This difference is because NV's explored region covers more area than AES at the beginning.

the cases where the labels are under (1) uniform noise and (2) Gaussian noise centered at the decision boundary.

We simulate the first case by randomly flipping the labels. The noisy label is set as $y' = (-1)^{\lambda} y$, where $\lambda \sim \text{Bernoulli}(p)$, p is the parameter of the Bernoulli distribution that indicates the noise level, and y is the true label.

The second case is probably more common in practice, since it is usually harder to decide the labels near the decision boundary. To simulate this case, we add Gaussian noise to the test functions: $g'(\boldsymbol{x}) = g(\boldsymbol{x}) + e$, where $g(\boldsymbol{x})$ is the Branin or Hosaki function, and $e \sim s \cdot \mathcal{N}(0, 1)$.

In each case, we compare the performance of AES ($\epsilon = 0.3, \eta = 1.3$) and NV (with tight bounds) under two noise levels. As expected, adding noise to the labels decreases the accuracy of both methods (Fig. 6.11a and 6.11b). However, in both cases (Bernoulli noise and Gaussian noise), the noise appears to influence NV more than AES. As shown in Fig. 6.12, when adding noise to the labels, NV has high error mostly along the input space boundaries, where it cannot query samples outside to further investigate those apparent feasible regions. In contrast, AES tries to exploit those rogue points to try to find new feasible regions, realizing after a few new samples that they are noise.

6.7.4 Effect of Dimensionality

To test the effects of dimensionality on AES's performance, we apply both AES and NV on higher-dimensional examples where the feasible domains are inside two ($d-1$)-spheres of radius 1 centered at $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively. Here $\boldsymbol{a} = \boldsymbol{0}$ and $\boldsymbol{b} = (3, 0, ..., 0)$. Fig. 6.13 shows the input space of the 3-dimensional double-sphere example. The initial point $\boldsymbol{x}^{(0)} = \boldsymbol{0}$. For the Gaussian process, we use a Gaussian kernel with a

length scale of 0.5. We set $\epsilon = 0.3$ and $\eta = 1.3$. To compute the F1 scores, we randomly generate 10,000 samples uniformly within the region where $x_1 \in [-2, 5]$ and $x_k \in [-2, 2], k = 2, ..., d$. The input space bounds for the NV algorithm are $x_1 \in [-1.5, 4.5]$ and $x_k \in [-1.5, 1.5], k = 2, ..., d$. We get the F1 scores and running time after querying 1,000 points.

As shown in Fig. 6.14, both AES and NV shows an accuracy drop and running time increase as the problem's dimensionality increases. This is expected, since based on the curse of dimensionality [14], the number of queries needed to achieve the same accuracy increases with the input space dimensionality. The curse of dimensionality is inevitable in machine learning problems. However, since AES explores the input space only when necessary (*i.e.*, only after it has seen the entire decision boundary of the discovered feasible domain), its queries do not need to fill up the large volume of the high-dimensional space. Therefore, AES's accuracy drop with problem dimensionality is not as severe as bounded methods like NV. For particularly high-dimensional design problems, another complementary approach is to construct explicit lower-dimensional design manifolds upon which to run AES [44, 49].

6.7.5 Nowacki Beam Example

To test AES's performance in a real-world scenario, we consider the Nowacki beam problem [157]. The original Nowacki beam problem is a design optimization problem where we minimize the cross-section area A of a cantilever beam of length l with concentrated load F at its end. The design variables are the beam's breadth b and height h. We turn this problem into a feasible domain identification problem by replacing the objective with a constraint $A = bh \leq 0.0025\text{m}^2$. Other constraints are (1) the maximum tip deflection $\delta = Fl^3/(3EI_Y) \leq 5\text{mm}$, (2) the

maximum bending stress $\sigma_B = 6Fl/(bh^2) \leq \sigma_Y$, (3) the maximum shear stress $\tau = 1.5F/(bh) \leq \sigma_Y/2$, (4) the ratio $h/b \leq 10$, and (5) the failure force of buckling $F_{crit} = (4/l^2)\sqrt{(GI_T)(EI_Z)/(1-\nu^2)} \geq fF$, where $I_Y = bh^3/12$, $I_Z = b^3h/12$, $I_T = I_Y + I_Z$, and f is the safety factor. And σ_Y, E, ν, and G are the yield stress, Young's modulus, Poisson's ratio, and shear modulus of the beam's material, respectively. We use the settings from [195], where $l = 0.5$m, $F = 5$kN, $f = 2$, $\sigma_Y = 240$MPa, $E = 216.62$GPa, $\nu = 0.27$, and $G = 86.65$GPa. As shown in Fig. 6.15, the feasible domain is a crescent-shaped region. Given only these constraints, it is unclear what appropriately tight bounds on the design variables should be.

In this experiment, we set the Gaussian kernel's length scale as 0.005, $\epsilon = 0.3$ and $\eta = 1.3$. The initial point $\boldsymbol{x}^{(0)} = (b_0, h_0) = (0.05, 0.05)$. The test samples are generated along a 100×100 grid in the region where $b \in [0, 0.02]$ and $h \in [0.1, 0.16]$.

After 242 iterations, the F1 score of AES reaches 0.933 and remains constant. Note that mostly the estimation error comes from the two sharp ends of the crescent-shaped feasible region (Fig. 6.15). This is because the kernel's assumption on function smoothness (*i.e.*, similar inputs should have similar outputs) causes the GP to have a bad performance where the labels shift frequently. The similar problem also exists when using other classifiers like SVM, where a kernel is also used to enforce similar outputs between similar inputs. This problem can be alleviated by using a smaller kernel length scale.

6.7.6 Detecting Novel Designs

We use two other real-world design examples to show how our proposed method discovers novel designs. Similar to the airfoil example, we represent each design with 100 Cartesian coordinates from their 2D outlines, and use the inverse mapping

Figure 6.16: Some of the initial designs used in the stemless glassware example.

$g : \mathcal{F} \in \mathbb{R}^2 \to \mathcal{X} \in \mathbb{R}^{200}$ generated by PCA to synthesize samples.

Different from previous examples, we use human judgment as the feasibility criterion. Although we can use some rules to determine the feasibility (*e.g.*, whether the outlines of a design are self-intersecting), a design's actual feasibility may depend on more complicated factors (*e.g.*, functionality and aesthetics). Therefore, initially we start with limited number and styles of real-world designs and do not have labels (*i.e.*, valid or invalid) for synthesized designs outside the real-world design domain until human assessment.

In the following two experiments, we select a sample in $\mathcal{F}$, and synthesize a design $\boldsymbol{x} \in \mathcal{X}$, where $\boldsymbol{x}$ consists of the Cartesian coordinates of that design's outline [49]. Then we use an interface that shows a human oracle the picture of that design, and then the oracle labels the design valid or invalid based on whether the designed shape is aesthetically pleasing. This process forms the function $h : \mathcal{F} \to \{-1, 1\}$. The specific choice of human oracle and interface is not central to the contributions of this chapter, serving only as an illustrative real-world example. In practice, we need to take human annotators' mistakes and disagreements into consideration. For example, we can model human disagreement by adding a white noise kernel to the Gaussian process kernel function. The white noise kernel is expressed as $k(\boldsymbol{x}, \boldsymbol{x}') = r$ if $\boldsymbol{x} = \boldsymbol{x}'$ and 0 otherwise, where r is the noise level. In the future, we will look into ways of dealing with the problem of input-dependent noise level (*i.e.*, different degree of human mistakes and disagreements for different design instances) [177, 193].

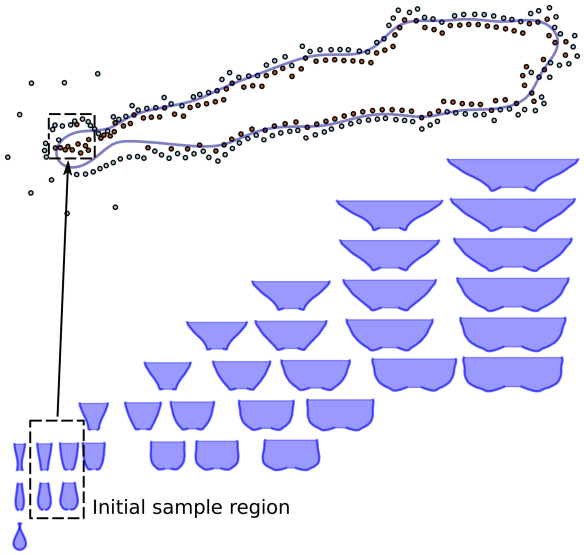


Figure 6.17: The discovered feasible domain and valid designs. The top figure shows the initial and queried samples and the estimated feasible domain in the embedding space $\mathcal{F}$ The solid dots represent valid designs, while the hollow dots represent invalid ones. Started with the stemless glasses shown in Fig. 6.16, the proposed method discovered other types of revolved objects such as vases and bowls.

Figure 6.18: Some of the initial designs used in the bottle example.

Example: stemless glasses as initial samples. In this example, we used only stemless glasses as initial designs (Fig. 6.16). We set the length scale of the Gaussian kernel as $l = 2.4$, and the margin $\epsilon = 1.5$. As shown in Fig. 6.17, on the given design manifold, our proposed method discovered a feasible domain with other revolved objects such as vases and bowls. The new designs increasingly differ as they get farther away from the initial design samples.

Example: bottles as initial samples. Similar to the previous example, we used only bottles as initial designs (Fig. 6.18). We set the length scale of the GP classifier $l = 1.3$, and the margin $\epsilon = 0.7$. As shown in Fig. 6.19, our proposed method

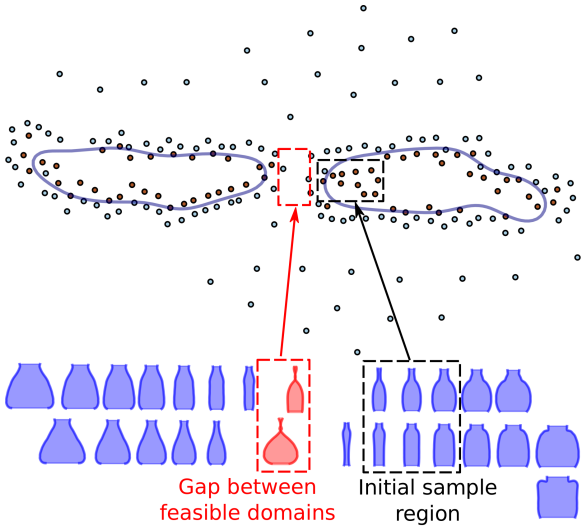


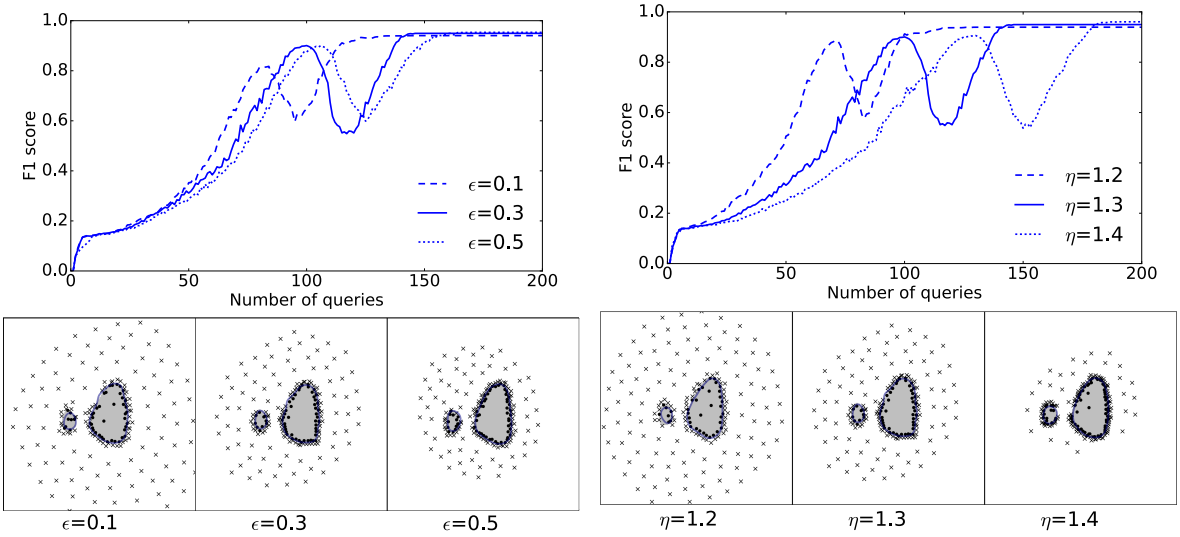
Figure 6.19: The discovered feasible domains and valid designs. The solid dots represent valid designs, while the hollow dots represent invalid ones. Started with the bottles shown in Fig. 6.18, the proposed method discovered two feasible domains, between which there are designs with self-intersecting contours.

discovered two feasible domains. In the left feasible domain, there are designs that look like bowling balls and flasks; and in the right feasible domain, there are designs that still look like bottles, but with a larger aspect ratio. Between these two feasible domains, designs have self-intersecting contours and thus are invalid.

6.8 Additional Experimental Results

6.8.1 Hosaki Example

We use the Hosaki example as an additional 2-dimensional example to demonstrate the performance of our proposed method. Different from the Branin example, the Hosaki example has feasible domains of different scales. Its feasible domains resemble two isolated feasible regions — a large "island" and a small one (Fig. 6.20a). The



(a) Changing ϵ ($\eta = 1.3$). (b) Changing η ($\epsilon = 0.3$).

Figure 6.20: AES with different ϵ and η on the Hosaki example.

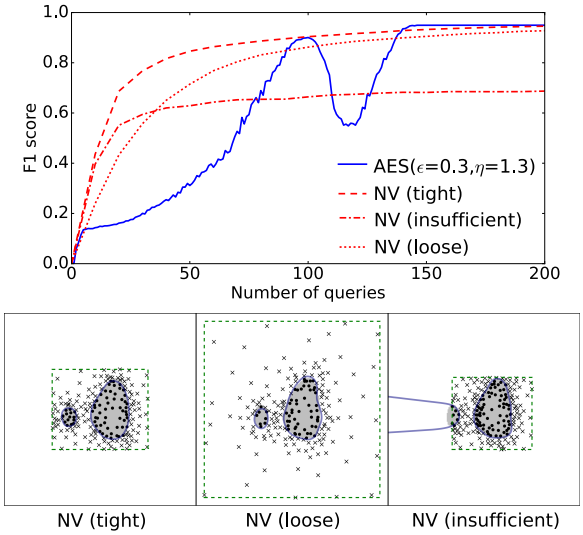
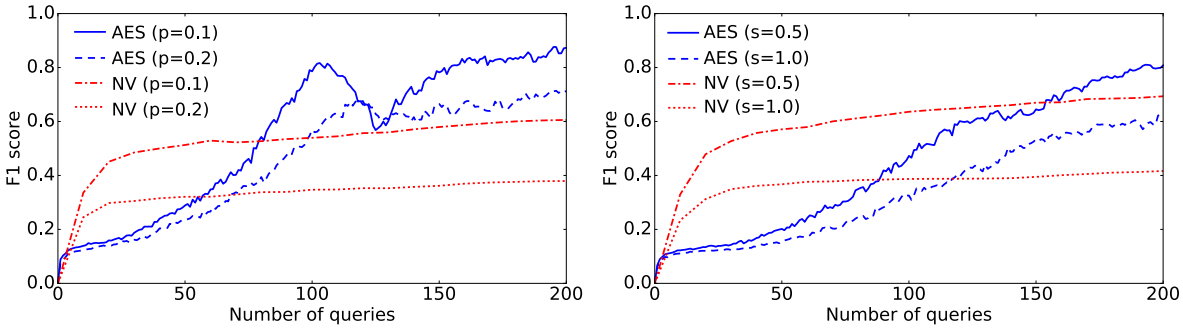


Figure 6.21: AES and NV (with different input variable bounds) on the Hosaki example.



(a) Bernoulli noise. (b) Gaussian noise.

Figure 6.22: AES and NV on the Hosaki example using noisy labels.

Table 6.3: Input space bounds for the NV algorithm and the straddle heuristic (Hosaki example).

	Tight	Loose	Insufficient
Hosaki	$x_1 \in [0, 6]$, $x_2 \in [0, 5]$	$x_1 \in [-2.5, 8.5]$, $x_2 \in [-3, 8]$	$x_1 \in [1, 6]$, $x_2 \in [0, 4.5]$

Hosaki function is

$$g(\boldsymbol{x}) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2}$$

We define the label $y = 1$ if $\boldsymbol{x} \in \{\boldsymbol{x} | g(\boldsymbol{x}) \leq -1, 0 < x_1, x_2 < 5\}$; and $y = -1$ otherwise.

For AES, we set the initial point $\boldsymbol{x}^{(0)} = (3, 3)$. We use a Gaussian kernel with a length scale $l = 0.4$. The test set to compute F1 scores is generated along a 100×100 grid in the region where $x_1 \in [-3, 9]$ and $x_2 \in [-3.5, 8.5]$. For NV and straddle, the input space bounds are shown in Tab. 6.3.

Table 6.4 shows the final F1 scores and running time of AES, NV, and the straddle heuristic. Fig. 6.20 shows the F1 scores and queries under different ϵ and η. Fig. 6.20. Fig. 6.21 compares the performance of AES and NV with different boundary sizes.

Table 6.4: Final F1 scores and running time (Hosaki example).

		F1 score	Time (s)
Hosaki (200 queries)	AES ($\epsilon = 0.3, \eta = 1.3$)	0.95 ± 0.003	28.25 ± 0.25
	AES ($\epsilon = 0.1, \eta = 1.3$)	0.94 ± 0.004	30.86 ± 0.19
	AES ($\epsilon = 0.5, \eta = 1.3$)	0.95 ± 0.002	28.32 ± 0.33
	AES ($\epsilon = 0.3, \eta = 1.2$)	0.94 ± 0.003	31.69 ± 0.45
	AES ($\epsilon = 0.3, \eta = 1.4$)	0.96 ± 0.002	26.39 ± 0.38
	NV (tight)	0.95 ± 0.003	22.58 ± 0.03
	NV (loose)	0.93 ± 0.004	22.28 ± 0.03
	NV (insufficient)	0.69 ± 0.010	22.27 ± 0.03
	Straddle (tight)	0.95 ± 0.002	16.20 ± 0.19
	Straddle (loose)	0.88 ± 0.005	14.00 ± 0.14
	Straddle (insufficient)	0.69 ± 0.010	16.92 ± 0.25

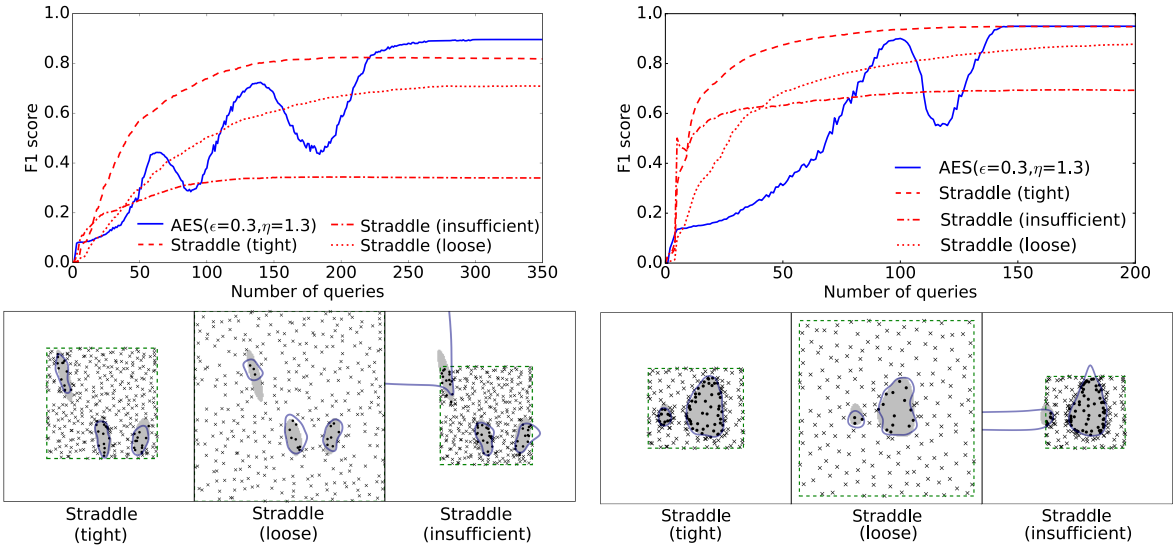
Fig. 6.22 shows the performance of AES and NV under Bernoulli and Gaussian noise.

6.8.2 Results of Straddle Heuristic

In this section, we list experimental results related to the straddle heuristic. Specifically, Fig. 6.23 shows straddle's F1 scores and queries using different sizes of input variable bounds, and the comparison with AES. Fig. 6.24 shows the comparison of AES and straddle under noisy labels.

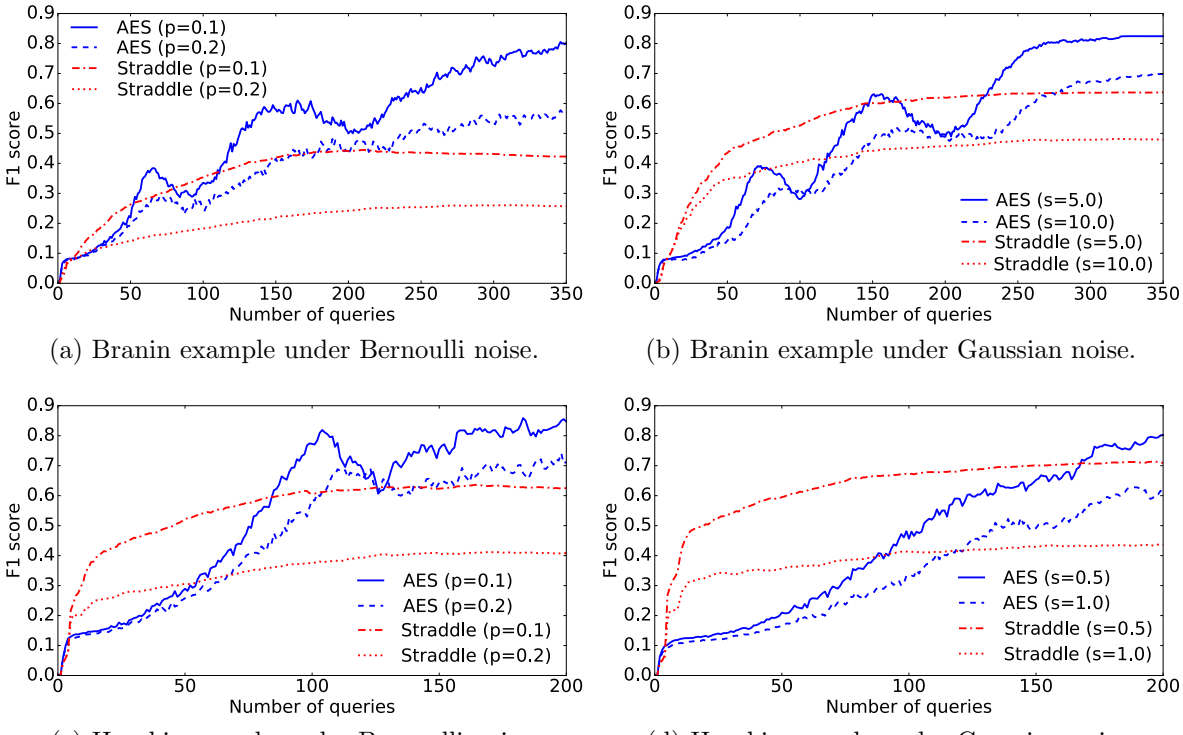
6.9 Summary

We presented a pool-based sampling method, AES, for identifying (possibly disconnected) feasible domains over an unbounded input space. Unlike conventional methods that sample inside a fixed boundary, AES progressively expands our knowledge of the input space under an accuracy guarantee. We showed that AES uses successive exploitation and exploration stages to switch between learning the decision boundary and searching for new feasible domains. To avoid increasing the pool



(a) Branin example. (b) Hosaki example.

Figure 6.23: AES and straddle (with different input variable bounds).



(a) Branin example under Bernoulli noise. (b) Branin example under Gaussian noise.

(c) Hosaki example under Bernoulli noise. (d) Hosaki example under Gaussian noise.

Figure 6.24: AES and straddle under noisy labels.

size and hence the computation cost as the explored area grows, we proposed a dynamic local pool generation method that samples the pool locally at a certain location in each iteration. We showed that at any point within the explored region, AES guarantees an upper bound ϵ of misclassification loss with a probability of at least $1 - \tau$, regardless of the number of iterations or labeled samples. This means that AES can be used for real-time prediction of samples' feasibility inside the explored region. We also demonstrated that, compared to existing methods, AES can achieve comparable or higher accuracy without needing to set exact bounds on the input space.

Note that AES cannot be applied to input spaces where synthesizing a useful sample is difficult. For example, in an image classification task, we cannot directly synthesize an image by arbitrarily setting its pixels, since most of the synthesized images may be unrealistic and hence useless. Usually in such cases, we use real-world samples as the pool and apply bounded active learning methods (since we know the bounds of real-world samples). Or instead, we first embed the original inputs onto a lower-dimensional space, such that given the low-dimensional representation, we can synthesize realistic samples. We can then apply AES on that embedded space. This approach can be used for discovering novel feasible domains (*i.e.*, finding feasible inputs that are nonexistent in the real-world). We refer interested readers to a detailed introduction of this approach by [44].

One limitation of AES is that the accuracy improves slowly at the early stage compared to bounded active learning methods. This is because AES focuses on only the explored region (which is small at the beginning), while bounded active learning methods usually do space-filling at first. In the situation where we want fast accuracy improvement at the beginning, one possible way of tackling this problem is by dynamically setting AES's hyperparameters. Specifically, since the expansion speed increases as ϵ or η decreases, we can accelerate AES's accuracy improvement

at earlier stages by setting small values of ϵ and η, so that queries quickly fill up a larger region. Then to achieve high final accuracy, we can increase ϵ and η to meet the accuracy requirement.

So far this dissertation has shown how to measure the intrinsic complexity of a design space to guide data-driven design synthesis, how to incorporate prior knowledge into data-driven design synthesis models, and how to identify feasible domains and discover novel designs when the design space or the latent space bounds are unclear. The next chapter will introduce a global optimization method that also expands an input space to allow the discovery of the optimal design being far away from existing designs.

Chapter 7: Global Optimization with Trust Region Bayesian Optimization

The work in this chapter has been submitted to the Conference on Neural Information Processing Systems (NeurIPS).

7.1 Introduction

Bayesian optimization (BO) is a global optimization technique targeted for expensive black-box functions [191]. Particularly, one of its important application in the machine learning community is automated hyperparameter tuning [201, 204, 207]. In a standard BO process, the objective function is modeled as a random function with a prior distribution. This prior updates to form a posterior after new observations (*i.e.*, a Gaussian process or GP [172]). The decision about which observation to collect next is made by globally maximizing an acquisition function based on the posterior. This step requires fixed variable bounds, which are sometimes not trivial to set. It is hard to guarantee that any fixed bounds will include the true global optimum.

In this chapter, we modified the standard BO approach so that the fixed variable bounds are not required. When the search space is unbounded, the acquisition function can have suprema at infinity, where the uncertainty is maximized. Thus we search only in the region with sufficiently low uncertainty, which we referred to as the *trust region* in reference to conceptually similar approaches from trust region

optimization. This trust region expands as we add more observations. We call this method Trust Region Bayesian Optimization (TRBO).

The main technical contributions of this chapter are:

1. A trust-region-based acquisition strategy to bound the GP model uncertainty and expand the search space; and

2. Theoretical results regarding how to adaptively set the threshold of the uncertainty bound to avoid the over-exploration problem that occurs in an expanding search space.

7.2 Bayesian Optimization

Bayesian Optimization uses a sequential strategy to search for the global optimum of expensive black-box functions. Assuming we have an objective function: $f : \mathbb{R}^d \to \mathbb{R}$, and the observation of its output has Gaussian noise: $y \sim \mathcal{N}(f(\mathbf{x}), \sigma_n^2)$. It is expensive to evaluate either the function f or its gradient (assuming we can only approximate the gradient by the finite difference method when f is a black-box function and that Automatic Differentiation methods cannot be used). Thus the goal of BO is to minimize the number of evaluations needed to find the global minimum. BO treats the objective as a random function that has a prior distribution, and update this prior to form a posterior distribution over the function after observing data. This can be done by using a Gaussian process (GP). The posterior distribution can then be used to form an acquisition criterion that proposes to evaluate f at a promising point, so that the regret is minimized. The GP posterior can then be updated after the new observation. This process repeats until the evaluation budget runs out or a satisfied solution is achieved. We will elaborate on the Gaussian process and the acquisition function in the following sections.

7.2.1 Gaussian Process

The Gaussian process (GP) [172] estimates the distribution of the objective function. A kernel (covariance) function $k(\mathbf{x}, \mathbf{x}')$ is used to measure the similarity between two points $\mathbf{x}$ and $\mathbf{x}'$. It encodes the assumption that "similar inputs should have similar outputs". The specific choice of kernel is not central to the core contributions of the chapter. In this chapter we use the RBF kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{\Lambda}^{-1}(\mathbf{x} - \mathbf{x}')\right) \tag{7.1}$$

where $\mathbf{\Lambda} = \text{diag}(l_1^2, ..., l_d^2)$ with l_i the length scale of the i-th dimension.

Given N observations $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i) | i = 1, ..., N\}$, the GP posterior $f(\mathbf{x})$ at any point $\mathbf{x}$ is a Gaussian distribution: $f(\mathbf{x})|\mathcal{D}, \mathbf{x} \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$ with the mean and the variance expressed as

$$\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \tag{7.2}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}(\mathbf{x}) \tag{7.3}$$

where $\mathbf{k}(\mathbf{x})$ is an N-dimensional vector with the i-th dimension being $k(\mathbf{x}, \mathbf{x}_i)$, and $\mathbf{K}$ is an $N \times N$ covariance matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

7.2.2 Acquisition Function

Bayesian optimization picks the next point to evaluate by maximizing an acquisition function, which is computed based on the GP posterior. Common acquisition functions include the probability of improvement (PI) [127], the expected improvement (EI) [108], the Gaussian Process upper confidence bound (GP-UCB) [205], and

those based on entropy search [93, 94, 229].

In this chapter, we use EI as our acquisition function. It measures the expected amount of improvement over the current best solution based on the learned GP model:

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) &= \mathbb{E}[\max\{0, f(\mathbf{x}) - f'\}] \\ &= \int_{f'}^{+\infty} (f - f')\mathcal{N}(f; \mu(\mathbf{x}), \sigma^2(\mathbf{x}))df \\ &= \sigma(\mathbf{x})(u\Phi(u) + \phi(u)) \end{aligned} \tag{7.4}$$

where f' is the current best objective function value, $u = (\mu(\mathbf{x}) - f')/\sigma(\mathbf{x})$, and Φ and ϕ are the cumulative density function (CDF) and probability density function (PDF) of the standard normal distribution, respectively.

7.2.3 Previous Work on Unbounded Bayesian Optimization

Normally Bayesian optimization is performed within fixed variable bounds. But in cases such as algorithm hyperparameter tuning [190, 204, 207] and shape optimization [164], setting the variable bounds are not trivial. It is hard to guarantee that any fixed bounds will include the true global optimum.

Two types of solutions were proposed to handle this problem: 1) performing BO in an unbounded space by regularization via non-stationary prior means so that the acquisition function's suprema will not be at infinity [170, 190]; and 2) performing BO in "soft bounds" that are gradually expanded over iterations [155, 156, 190]. The first solution computes an acquisition function that is biased toward regions near some user-specified center point, thus insufficient exploitation may occur when the optimal solution is far from the center. The second solution either expands each direction equally, which often yields to unnecessarily large search spaces [170, 190]; or expands only to the promising region where the upper confidence bound (UCB) is

larger than the lower confidence bound (LCB) of the current best solution [155, 156]. But the latter approach has to perform global search twice at each iteration—one for the maximum LCB to filter out the non-promising region; and one for the maximum acquisition function value. It expands the bounds of the first search according to a hard-coded rule, and hence may show a lack of adaptability to different optimization problems.

The trust region Bayesian optimization is different from the previous methods in that it adaptively expands the search space based on the uncertainty of the GP model. It can essentially avoid the aforementioned issues by employing a strategy that we will introduce in the following section.

7.3 Trust Region Bayesian Optimization

In this section we will introduce the main ingredients of TRBO, namely, its acquisition strategy (Sect. 7.3.1), global optimization of the acquisition function (Sect. 7.3.2), the way of adaptively balancing exploration and exploitation (Sect. 7.3.3), and a trick to improve exploitation when expanding the search space (Sect. 7.3.4).

7.3.1 Acquisition Strategy

Our acquisition strategy can be expressed as the following constrained optimization problem:

$$\begin{aligned} &\max_{\mathbf{x}\in\mathbb{R}^d} \ \mathrm{EI}(\mathbf{x}) \\ &\text{s.t. } \sigma^2(\mathbf{x}) \leq \tau k_0 \end{aligned} \tag{7.5}$$

where $\tau \in (0, 1)$ is a coefficient controlling the aggressiveness/conservativeness of exploration, and $k_0 = \sigma^2(\mathbf{x}_\infty)$ with $\mathbf{x}_\infty$ denotes a point infinitely far away from the observations. Based on Eq. 7.3, we have $k_0 = \sigma^2(\mathbf{x}_\infty) = k(\mathbf{x}, \mathbf{x})$. When using the

kernel shown in Eq. 7.1, simply we have $k_0 = 1$. Under this acquisition strategy, only points with low GP model uncertainty will be picked for evaluation. We also included the acquisition strategy for the constrained BO setting in the supplementary material.

To avoid excessive local exploitation, we can modify the equation for EI (Eq. 7.4) as

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}[\max\{0, f(\mathbf{x}) - (f' + \epsilon)\}] \tag{7.6}$$

where $\epsilon > 0$ is the *minimum improvement parameter* [107, 189, 190].

In many real-world cases, Bayesian optimization needs to deal with constrained problems of the following two kinds: 1) there are infeasible regions in the input space (*e.g.*, some experimental configurations are infeasible); and 2) the objective function f does not have definition in some regions of the input space (*e.g.*, when the hyperparameters of a neural network are not properly chosen, exploding gradients may occur, which may lead to NaN weight values and hence the NaN accuracy). This is especially common when the we have an unbounded or expanding search space. Therefore, it is worth extending TRBO to make it suitable for constrained problems. Specifically, we can modify Eq. 7.5 based on Refs. [11] and [78]:

$$\begin{aligned} \max_{\mathbf{x}\in\mathbb{R}^d} \ & \mathrm{EI}(\mathbf{x})\Pr(\mathcal{C}(\mathbf{x})) \\ \text{s.t. } & \sigma^2(\mathbf{x}) \leq \tau k_0 \\ & \Pr(\mathcal{C}(\mathbf{x})) \geq 0.5 \end{aligned} \tag{7.7}$$

where $\mathcal{C}(\mathbf{x})$ is an indicator of whether the constraints are satisfied or whether the objective function has definition.

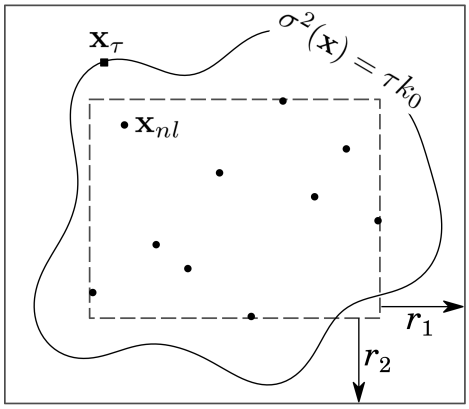


Figure 7.1: Feasible domain bounds. In each iteration, we expand the minimum bounding box of evaluated samples along the i-th axis by r_i.

7.3.2 Feasible Domain Bounds

The feasible domain defined by Eq. 7.5 is bounded by the isocontour $\sigma^2(\mathbf{x}) = \tau k_0$. However, it is easier to search inside a bounding box instead of an irregular isocontour when solving the global optimization problem in Eq. 7.5. We can show that the solution to Eq. 7.5 is inside a bounding box, which we call the *feasible domain bounds*. In this section, we will derive the feasible domain bounds.

For any point $\mathbf{x}_\tau$ on the isocontour (Fig. 7.1), *i.e.*, $\sigma^2(\mathbf{x}_\tau) = \tau k_0$, based on Eq. 7.3 we have $k_0 - \mathbf{k}_\tau^\top \mathbf{A} \mathbf{k}_\tau = \tau k_0$, or

$$\mathbf{k}_\tau^\top \mathbf{A} \mathbf{k}_\tau = (1 - \tau) k_0 \tag{7.8}$$

where $\mathbf{A} = (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}$ and $\mathbf{k}_\tau = \mathbf{k}(\mathbf{x}_\tau)$.

Since $\mathbf{A}$ is symmetric, we have $\lambda_{\min} \mathbf{k}_\tau^\top \mathbf{k}_\tau \leq \mathbf{k}_\tau^\top \mathbf{A} \mathbf{k}_\tau \leq \lambda_{\max} \mathbf{k}_\tau^\top \mathbf{k}_\tau$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest eigenvalue of $\mathbf{A}$, respectively. Thus $\mathbf{k}_\tau^\top \mathbf{k}_\tau$ have the following bounds for any $\mathbf{x}_\tau$:

$$(1 - \tau) k_0 / \lambda_{\max} \leq \mathbf{k}_\tau^\top \mathbf{k}_\tau \leq (1 - \tau) k_0 / \lambda_{\min} \tag{7.9}$$

Suppose $\mathbf{x}_{nl}$ is the nearest evaluated point to $\mathbf{x}_\tau$ (Fig. 7.1), the following inequality holds:

$$\mathbf{k}_\tau^\top \mathbf{k}_\tau < N k^2(\mathbf{x}_{nl}, \mathbf{x}_\tau) \tag{7.10}$$

where $N > 1$ is the number of evaluated points.

According to Eq. 7.9 and Eq. 7.10, we have

$$N k^2(\mathbf{x}_{nl}, \mathbf{x}_\tau) > (1-\tau) k_0 / \lambda_{\max} \tag{7.11}$$

for any $\mathbf{x}_\tau$. Also, based on Eq. 7.1, we have

$$k^2(\mathbf{x}_{nl}, \mathbf{x}_\tau) = \exp\left(-\sum_{i=1}^{d} \left(\frac{\delta_i}{l_i} \right)^2 \right) \tag{7.12}$$

where $\delta = |\mathbf{x}_{nl} - \mathbf{x}_\tau|$. Substituting Eq. 7.12 into Eq. 7.11, we get the following inequality

$$\sum_{i=1}^{d} \frac{\delta_i^2}{C l_i^2} < 1 \tag{7.13}$$

where $C = -\log((1-\tau)k_0/(N\lambda_{\max}))$. Equation 7.13 shows that $\mathbf{x}_\tau$ is inside a d-dimensional hyperellipsoid that centered at $\mathbf{x}_{nl}$ with

$$r_i = \sqrt{C} l_i = l_i \sqrt{-\log((1-\tau)k_0/(N\lambda_{\max}))} \tag{7.14}$$

corresponding to half the length of the i-th principal axis. Thus by setting the bounds of the i-th dimension as $\left[\min_j\{x_i^{(j)}\} - r_i, \max_j\{x_i^{(j)}\} + r_i\right]$, we can include the entire feasible domain of Eq. 7.5. This means that in each iteration, we get the minimum bounding box of all evaluated samples, and expand the bounding box along the i-th axis by r_i (Fig. 7.1). Then constrained global optimization of the acquisition function is performed within the new bounds.

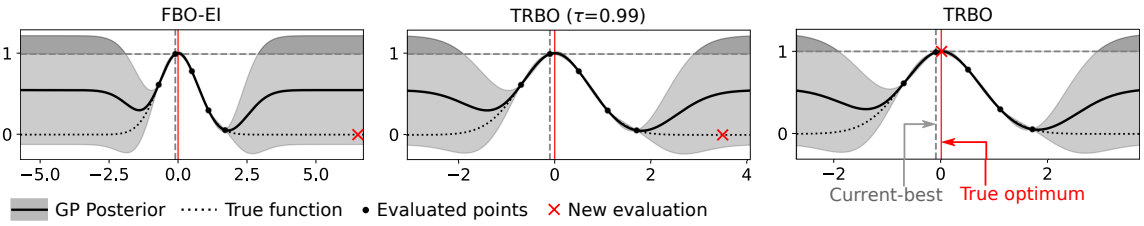


Figure 7.2: Over-exploration in FBO-EI (left) and TRBO with large τ (middle); By adaptively setting τ, TRBO enforces exploitation based on an accuracy criterion (right).

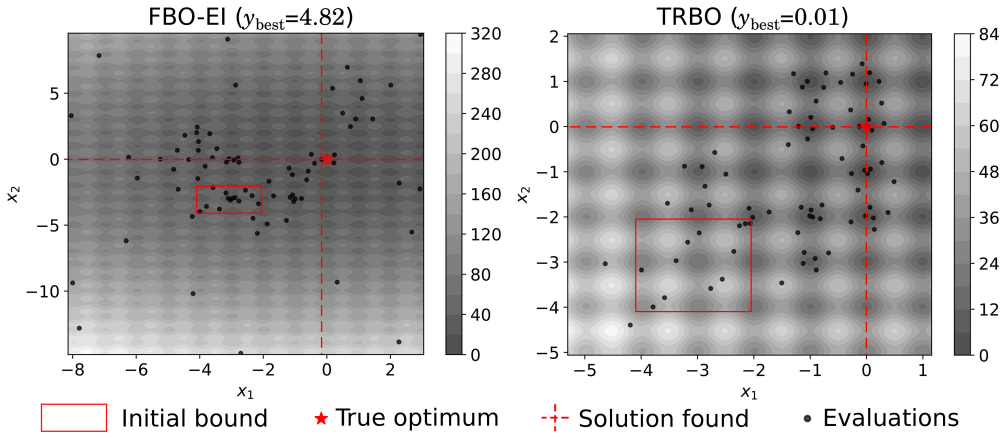


Figure 7.3: In contrast with TRBO (right), FBO-EI (left) spends too much budget on randomly exploring the search space (left).

In practice, because Eq. 7.10 is usually quite loose (especially when N is large), the above derived bounds are usually unnecessarily large, causing large volume of infeasible domain inside the bounds. In that case, we can replace $\lambda_{\max}$ with $\lambda_{\min}$ in Eq. 7.11, *i.e.*, substituting the upper bound of $\mathbf{k}_\tau^\top \mathbf{k}_\tau$ (Eq. 7.9) into Eq. 7.11.

7.3.3 Adaptive Exploration-Exploitation Trade-off

A problem of an expanding search space is that a new evaluation may get too far away from the region of interest, due to the high uncertainty and hence the high EI in far-away regions (Fig. 7.2). The informativeness of those high-uncertainty regions, however, is low because of the sparsity of observed data near them. Thus sampling at

those regions is like shooting in the dark. This is fine in the fixed-bound BO, because it will exploit the region of interest eventually after finishing exploring those high-uncertainty regions. However, with expanding bounds, the high-uncertainty regions are expanding and BO could continuously sample in those regions and never head back to exploit the region of interest. As a result, the algorithm will spend too much budget on randomly exploring the search space but have insufficient exploitation, as shown in the left plot of Fig. 7.3. We call this *over-exploration*.

This over-exploration problem can exist in every unbounded Bayesian optimization algorithm with an aggressive expansion strategy. Trust region BO can solve this problem by avoiding exploring in regions where our estimated model is uncertain (*i.e.*, constraining the GP's predictive variance $\sigma^2(\mathbf{x})$, see Eq. 7.5). However, one has to choose a proper coefficient τ to set the uncertainty threshold. In this section, we derive a way of setting τ adaptively to balance exploration and exploitation as the search space expands.

The simplest way to avoid over-exploration is to force the algorithm to stop exploring and start to refine the solution by exploiting near the current best point. In TRBO, exploration is performed by sampling points along the feasible domain boundary (*i.e.*, $\sigma^2(\mathbf{x}) = \tau k_0$). Thus, we can avoid over-exploration by decreasing τ so that the expected improvement on the boundary is lower than that near the current best solution $\mathbf{x}'$.

The expected improvement on the boundary can be expressed via the predictive mean μ_τ:

$$\mathrm{EI}_\tau(\mu_\tau) = (\mu_\tau - f')\Phi\left(\frac{\mu_\tau - f'}{\sqrt{\tau k_0}}\right) + \sqrt{\tau k_0}\phi\left(\frac{\mu_\tau - f'}{\sqrt{\tau k_0}}\right) \tag{7.15}$$

Also we have

$$\max_{\sigma^2(\mathbf{x})=\tau k_0} \{\mathrm{EI}_\tau(\mu_\tau)\} = \mathrm{EI}_\tau\left(\max_{\sigma^2(\mathbf{x})=\tau k_0} \{\mu_\tau\}\right) = \mathrm{EI}_\tau(\mu_m) \tag{7.16}$$

since EI monotonically increases with the predictive mean.

The expected improvement near the current best solution $\mathbf{x}'$ is

$$\mathrm{EI}_+ = (\mu_+ - f')\Phi\left(\frac{\mu_+ - f'}{\sigma_+}\right) + \sigma_+\phi\left(\frac{\mu_+ - f'}{\sigma_+}\right) \tag{7.17}$$

where μ_+ and σ_+ are the predictive mean and standard deviation respectively at a point $\mathbf{x}_+$ near the current best solution. Assuming that the GP mean function $\mu(\mathbf{x})$ is Lipschitz continuous, we have $f' - \mu_+ = \delta$, where δ is a small positive real number. Thus we have $u_+ = -\delta/\sigma_+$.

Now we can set $\mathrm{EI}_+ > \mathrm{EI}_\tau(\mu_m)$ to encourage exploitation. However, we do not want pure exploitation. Specifically, we want to stop exploitation at $\mathbf{x}_+$ whenever the *room for improvement* over the current solution f' within the neighborhood of $\mathbf{x}'$ is sufficiently low with a high probability:

$$\Pr(f_+ - f' \leq \xi) \geq 1 - \kappa \tag{7.18}$$

where $f_+ \sim \mathcal{N}(\mu_+, \sigma_+^2)$, and $\xi \geq 0$ and $0 < \kappa < 1$ are small real numbers. From Eq. 7.18 we can derive

$$\sigma_+ \leq \frac{\xi + \delta}{\Phi^{-1}(1-\kappa)} = \sigma_0$$

Thus we only need to exploit at $\mathbf{x}_+$ when $\sigma_+ > \sigma_0$. By substituting it into Eq. 7.17,

we get a lower bound for EI_+:

$$\mathrm{EI}_+ > -\delta\Phi(-\delta/\sigma_0) + \sigma_0\phi(-\delta/\sigma_0) = \mathrm{EI}_0 \tag{7.19}$$

since EI monotonically increases with the predictive variance. We can set this lower bound EI_0 equal to $\mathrm{EI}_\tau(\mu_m)$ to enforce $\mathrm{EI}_+ > \mathrm{EI}_\tau(\mu_m)$ when $\sigma_+ > \sigma_0$ (*i.e.*, when exploitation is necessary). Using Eq. 7.15, we can write $\mathrm{EI}_\tau(\mu_m) = \mathrm{EI}_0$ as

$$(\mu_m - f')\Phi\left(\frac{\mu_m - f'}{\sqrt{\tau k_0}}\right) + \sqrt{\tau k_0}\phi\left(\frac{\mu_m - f'}{\sqrt{\tau k_0}}\right) = \mathrm{EI}_0 \tag{7.20}$$

We can solve for τ by using any root finding algorithm (*e.g.*, Newton's method).

At the beginning of the optimization process, we do not need to make sure the room for improvement over f' is small within the neighborhood of $\mathbf{x}'$. Rather, we want to explore other regions that may contain better local optima. Thus we can set $\xi = \xi_0$ at the beginning, where a larger ξ_0 allows more exploration, and then anneal ξ over iterations until $\xi = 0$ (*e.g.*, towards the end of a computational budget). As a result, TRBO's focus gradually switches from exploration to exploitation.

In practice, we can set μ_m as the prior mean (0 by default), since it is usually the case when over-exploration occurs. Thus this adaptive approach can effectively avoid over-exploration, as shown in Fig. 7.2 and Fig. 7.3. If in reality $\mu_m < 0$, then $\mathrm{EI}_\tau(\mu_m) < \mathrm{EI}_0 < \mathrm{EI}_+$, TRBO will exploit near the current best solution even when it is unnecessary (*i.e.*, $\sigma_+ < \sigma_0$); while if $\mu_m > 0$, then $\mathrm{EI}_\tau(\mu_m) > \mathrm{EI}_0$, TRBO may explore when it should exploit.

7.3.4 Local Search for Better Exploitation

In practice, to perform the global optimization of Eq. 7.5, we can sample initial *candidate solutions* within the bounds derived in Sect. 7.3.2, and refine those solutions using a gradient-based optimization method (*e.g.*, L-BFGS-B [29]). If the GP kernel is fixed, the search space is always expanding, because σ is monotonically non-increasing as the number of observation increases. Specifically, the space near the queried point will be added to the search space volume. This results in a volume increase that is exponential with respect to the search space dimensionality. It will become harder for the candidate solutions to maintain the coverage of the search space as the optimization proceeds, especially when the problem has high dimensionality. Although we keep increasing exploitation by annealing ξ, it does not guarantee that we will exploit near the current best solution in a large search space. This problem was not addressed in previous unbounded Bayesian optimization methods [155, 190]. A straightforward way to solve the problem is to increase the density of search algorithms, but this continuously increases the computational cost for each iteration. Alternatively, we propose local search near the current best solution to allow better exploitation. Specifically, in each iteration, we generate the same number of candidate solutions but divide it for two tasks—global search and local search. Global search tries to find a promising point in the entire feasible domain in Eq. 7.5; while local search tries to find a promising point near the current best solution. This avoids insufficient exploitation but will not increase the computational cost.

The optimization process is summarized in Algorithm 4.

Algorithm 4 Trust region Bayesian optimization

▷ Given objective function f, initial bounds $\mathcal{B}$, initial evaluation n, and evaluation budget N
procedure OPTIMIZE($f, \mathcal{B}, n, N$)
    Sample n points $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ in $\mathcal{B}$ using LHS
    $y_i \leftarrow f(\mathbf{x}_i)$, $\forall i = 1, ..., n$
    $\mathcal{D} \leftarrow \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
    for $t = (n + 1) : N$ **do**
        Fit the GP model $\mathcal{M}$ to $\mathcal{D}$
        Compute τ based on Eq. 7.20
        Expand the minimum bounding box of $\mathcal{D}$ (Eq. 7.14) as the feasible domain bounds $\mathcal{B}'$
        Search for the solution $\mathbf{x}_t$ to Eq. 7.5 or Eq. 7.7 inside $\mathcal{B}'$
        $y_t \leftarrow f(\mathbf{x}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(\mathbf{x}_t, y_t)\}$
    end for
end procedure

7.4 Experiments

We evaluate the TRBO on both a range of synthetic test functions and an MLP hyperparameter optimization task. We also demonstrate the effect of dimensionality on TRBO's performance and the significance of adaptive exploration-exploitation trade-off in TRBO.

7.4.1 Experimental Protocol

The evaluation budget was set to $50d$, and an initial sample size of $5d$ was drawn by using Latin hypercube sampling [108, 149]. For simplicity we used an isotropic kernel for the GP (*i.e.*, $l_1 = ... = l_d$). We normalized the observed function outputs before fitting a GP regression model. We set $\xi_0 = 0.1$, $\kappa = 0.1$, $\epsilon = 0.01$, and $\delta = 0.01$. For each test function, we set the initial bounds to be [10%, 30%] of its original bounds, as was also configured in Ref. [155]. All the initial bounds do not

Table 7.1: Optimization results for synthetic benchmarks

Method	SixHumpCamel	Branin	Rastrigin	Hartmann3
TRBO	$\mathbf{-1.03 \pm 0.00}$	$\mathbf{0.41 \pm 0.01}$	$\mathbf{0.21 \pm 0.39}$	$\mathbf{-3.74 \pm 0.17}$
FBO-EI	-0.97 ± 0.05	1.27 ± 1.49	4.52 ± 2.29	-1.50 ± 0.93
FBO-UCB	-0.85 ± 0.09	1.43 ± 0.61	5.50 ± 2.60	-2.31 ± 1.34
EI-Q	-0.28 ± 0.37	2.95 ± 1.73	8.10 ± 1.47	-2.43 ± 0.65
EI-H	-0.47 ± 0.46	1.89 ± 1.00	7.39 ± 1.29	-3.41 ± 0.25

Method	Hartmann6	Beale	Rosenbrock
TRBO	-3.30 ± 0.04	$\mathbf{0.26 \pm 0.29}$	$\mathbf{0.44 \pm 0.40}$
FBO-EI	$\mathbf{-3.30 \pm 0.03}$	0.41 ± 0.32	7.72 ± 9.25
FBO-UCB	-3.26 ± 0.04	0.46 ± 0.37	17.39 ± 33.04
EI-Q	-2.32 ± 0.23	4.25 ± 2.83	17.45 ± 20.12
EI-H	-2.82 ± 0.15	3.87 ± 3.31	20.63 ± 19.71

include global optima. We compared TRBO to methods from Ref. [190] (*i.e.*, EI-Q and EI-H) and Ref. [155] (*i.e.*, FBO-EI and FBO-UCB).

7.4.2 Synthetic Benchmarks

We used seven standard global optimization test functions. As shown in Table 7.1, TRBO out-performs other methods on most test functions. Note that TRBO's results have lower variance compared to other methods, which is an indication of robustness. Figure 7.4 shows the optimization history on benchmark functions. It shows that compared to the other two state-of-the-art methods, trust region Bayesian optimization (TRBO) converged faster and achieved a better solution in most cases.

We also demonstrated the effects of problem dimensionality on TRBO's performance by using two synthetic benchmarks, as shown in Fig. 7.5. Here we define the *optimality gap* $e = y_{sol} - y_{opt}$, where y_{sol} and y_{opt} are the minimal observation and the true minimum of the objective function, respectively. We compared TRBO to: 1) the other two state-of-the-art methods—FBO-EI and EI-H, and 2) the standard BO with

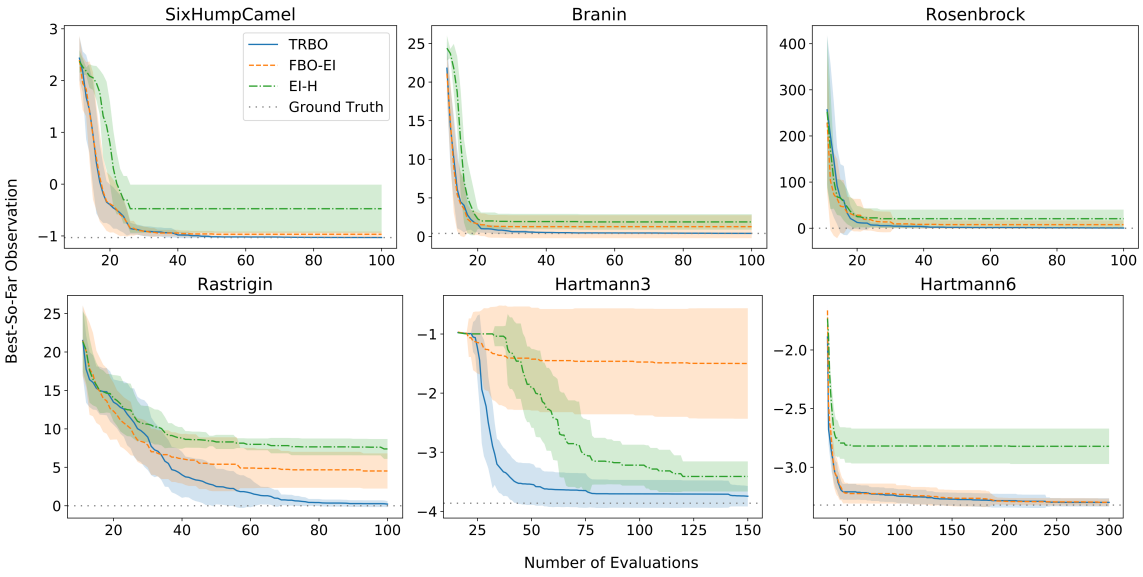


Figure 7.4: Optimization history for synthetic benchmarks

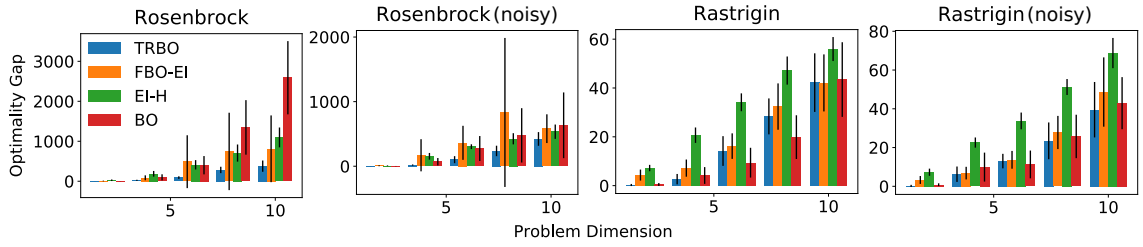


Figure 7.5: The effect of problem dimension on optimality gaps.

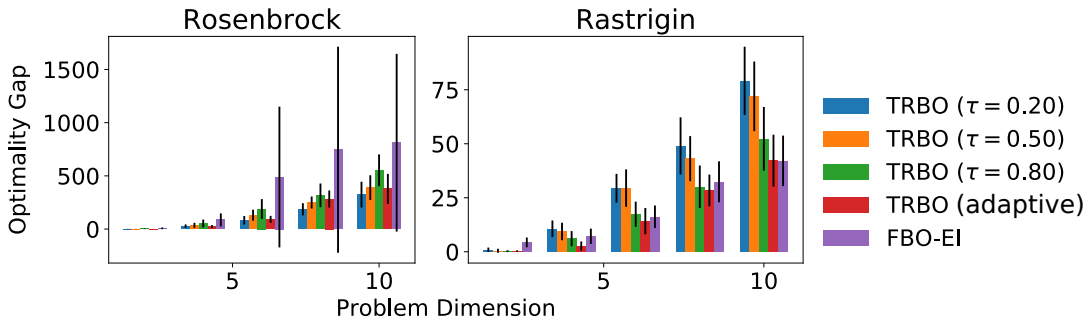


Figure 7.6: The effect of fixed and adaptive τ on optimality gaps. Here FBO-EI is shown as a baseline.

the original function bounds (which include global optima). We also tested the case where the observations are corrupted by Gaussian noise with a standard deviation of 0.1. The results show that the optimality gap increases with the problem dimension, which can be explained by the curse of dimensionality [14]. TRBO demonstrated the best performance among methods dealing with an unbounded or expanding search space, and is almost as good as the standard BO on the Rastrigin function. Since the global optimum of the Rosenbrock function is inside a narrow flat valley, it is trivial to find the valley but difficult to converge to the global optimum. Thus it requires large budget for exploitation in that valley (*i.e.*, *exploitation-intense*). Methods like FBO-EI, EI-H, and the standard BO may have unnecessarily large search space and hence waste budget on exploring regions far from the global optimum, rather than exploiting the valley. Thus compared to TRBO, it is more difficult for these three methods to find good solutions on the Rosenbrock function, especially when the dimensionality is high (Fig. 7.5).

To demonstrate the effectiveness of adaptive exploration-exploitation trade-off, we ran TRBO with both fixed τ and adaptive τ solved from Eq. 7.20. The two test functions, Rosenbrock and Rastrigin, have different characteristics and hence prefer different exploration-exploitation trade-offs. Since the Rastrigin function has

a large number of local optima, the difficulty for optimizing on Rastrigin is to avoid getting stuck in those local optima. Thus an algorithm with a higher search space expanding rate (*e.g.*, TRBO with a *large* τ) is likely to perform better since it will spend less budget exploiting local optima and more budget expanding towards the global optimum (*i.e.*, *exploration-intense*). In contrast, due to the narrow flat valley in the Rosenbrock function, the difficult part is exploiting near the global optimum to refine the solution. Thus a lower expanding rate (*e.g.*, TRBO with a *small* τ) is likely to be preferred since less budget will be wasted for exploration. The results shown in Fig. 7.6 are consistent with our expectation: the optimality gap increases with the value of the fixed τ on the Rosenbrock function, while the opposite behavior was observed on the Rastrigin function. However, by using an adaptive τ, TRBO performs better than most other configurations on both test functions. Note that the behavior of FBO-EI is similar to TRBO with a large τ (without considering the high performance variance on the Rosenbrock function).

As every objective function weights exploitation and exploration differently, BO methods with a fixed expansion schedule may succeed for one function, but fail for another. The TRBO can avoid this by adaptively balancing exploitation-exploration while expanding the search space.

7.4.3 Constrained Problems

We created two test problems to evaluate the performance of TRBO in dealing with constrained problems. Specifically, the constrained Rastrigin problem uses the Rastrigin function as the objective function, and the feasible domain is defined by an ellipse $0.01x_1^2 + (x_2 + 2)^2 \leq 1$ (Fig. 7.7). The Nowacki beam problem is a real-world test problem originally described by Nowacki [157, 195]. The goal is to minimize the

cross-sectional area of a tip-loaded cantilever beam subject to certain constraints.[1]

The results of the two problems are shown in Figs. 7.7 and 7.8. For the constrained Rastrigin problem, TRBO achieved a better solution than the other two methods. For the Nowacki beam problem, FBO-EI's solution has the lowest mean value but a very high variance; while TRBO found a fairly close optimal solution with much lower variance. The evaluated points by TRBO were dense near optima (either local or global). This behavior was, however, not obvious for the other two methods. This is likely because that FBO-EI and EI-H over-trusted the GP posterior even where its uncertainty was high. This resulted in sampling patterns with too much randomness, and hence higher variance of optimal solutions.

7.4.4 MLP on MNIST

We use the hyperparameter optimization of a multilayer perceptron (MLP) as a real-world example to demonstrate the performance of the TRBO. MNIST was used as the training data. The MLP has 512 hidden units with ReLU activations and was implemented using TensorFlow [1]. We used Adam [116] as the MLP's optimizer. We optimized seven hyperparameters, namely the learning rate, the learning rate decay, the dropout rate, and the L1 and L2 regularization coefficients for each layer. We performed the optimization in the log space (base 10) with the initial bounds of $[-5, -4]^7$. The objective is to maximize the accuracy of the MLP. The observed accuracy values are usually left-skewed, because usually there are a few very bad values initially sampled by LHS, after which most observations have high accuracy. Thus we apply a cube transformation before normalizing the accuracy values. As

[1]The original problem is a multi-objective optimization problem that minimizes both the cross-sectional area and the bending stress. Here we only consider the first objective and limit the second objective (*i.e.*, the bending stress should be smaller than the yield stress of the material) to form an extra constraint.

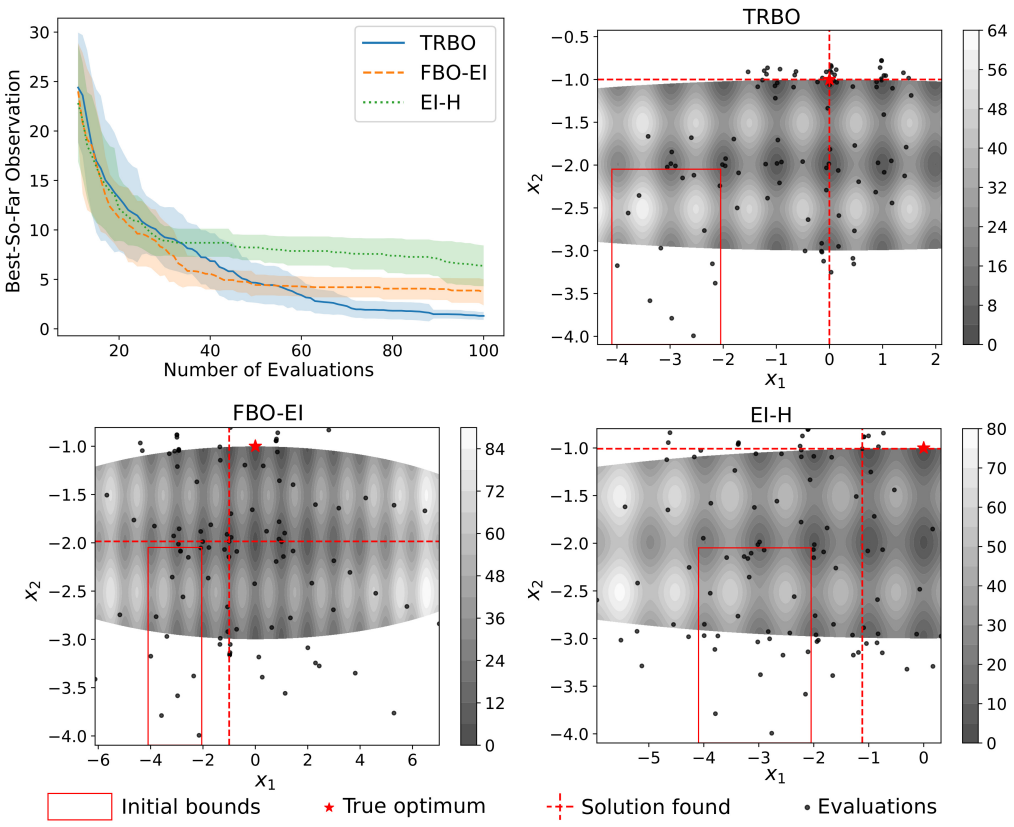


Figure 7.7: Optimization history and evaluated points for the constrained Rastrigin problem

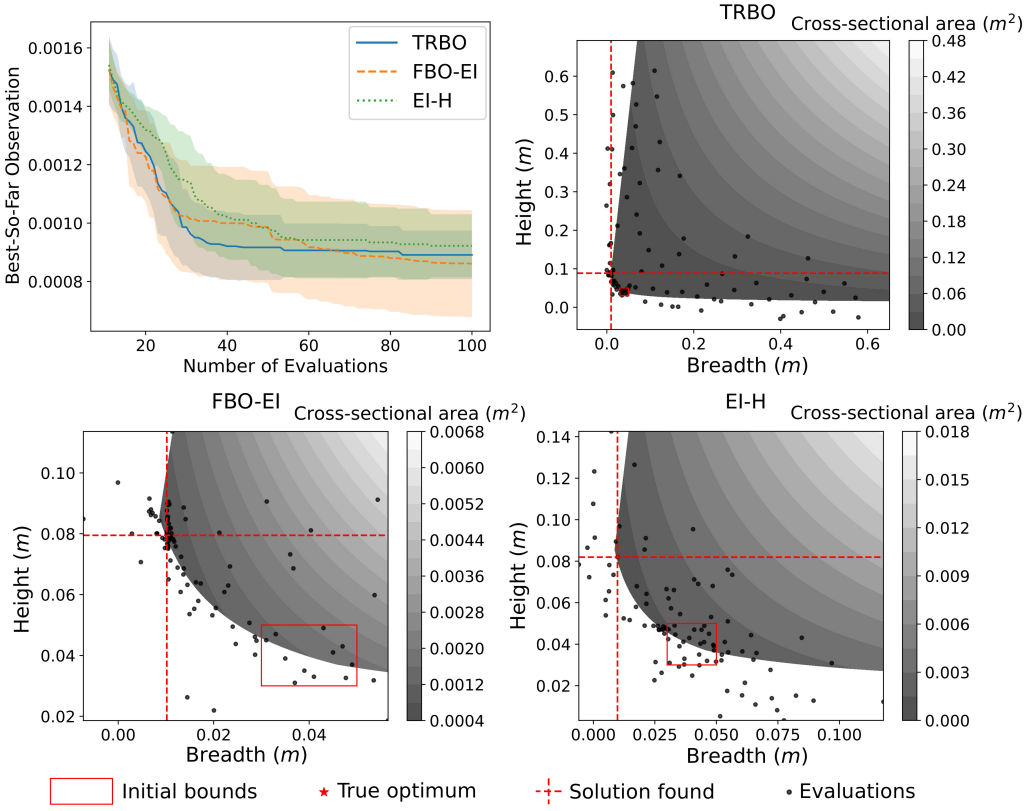


Figure 7.8: Optimization history and evaluated points for the Nowacki beam problem

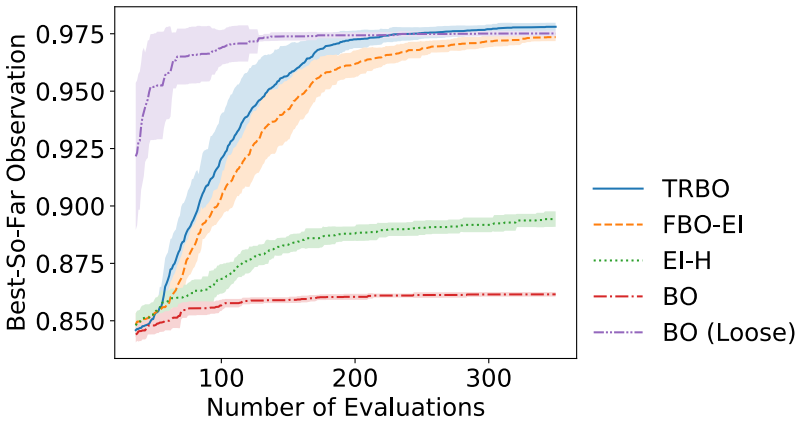


Figure 7.9: Optimization history for the hyperparameter tuning of a MLP trained on MNIST.

shown in Fig. 7.9, TRBO converged faster and found a better optimal solution than FBO-EI and EI-H.

7.5 Summary

We proposed a Bayesian optimization method, TRBO, that gradually expands the search space, so that we can find the global optimum without having to specify the input space bounds that include it. The proposed method only evaluates samples at regions with low GP model uncertainty, and expands the search space adaptively to avoid over-exploration in an expanding search space. This method is useful in cases where we are not confident about the range of the global optimum. The experimental results show that our method outperforms the other state-of-the-art methods in most cases.

In the standard BO, even if the input space bounds are set large enough to cover the global optimum, too much budget may be spent on needlessly exploring the large space. This will result in bad solutions when optimizing an exploitation-intense objective function, as shown by the Rosenbrock example.

So far this dissertation has shown how to measure the intrinsic complexity of a design space to guide data-driven design synthesis, how to incorporate prior knowledge into data-driven design synthesis models, and how to explore design spaces or latent spaces when their bounds are unclear. The next chapter will summarize these contributions, look at their broader implications, and discuss possible future research directions.

Chapter 8: Conclusion

Many traditional design processes go through the cycle of ideation, prototyping, evaluating, and improving. These processes involve human intervention, and can be expensive and time-consuming. Given a limited budget, only a few designs can be implemented and tested, and the true optimal design is hard to uncover since insufficient design alternatives are explored. Currently, topology optimization can be used to avoid human intervention in the traditional design process and automatically generate suboptimal designs. However, it does not reveal the underlying properties of a design space—*e.g.*, what is the least number of factors that are sufficient to control the shape variation, how does the shape change in the design space, and what is the global optimal solution to a design problem.

This dissertation addressed some of these related problems in traditional design processes and emergent generative design approaches by learning a compact representation from data. The compact representation encodes important properties of the design space and makes it less expensive to search for desired designs. This chapter summarizes the main ideas in this dissertation, discusses some broader implications for the whole body of work, and proposes some potential future research directions.

8.1 Dissertation Summary

This dissertation first reviewed problems in traditional design processes, including repetitive and expensive human intervention. Parametric design synthesis and design space exploration use consistent parameterization and automated algorithms to avoid human intervention, but still, the high-dimensional design space is difficult to understand and search for solutions. To solve these problems, this dissertation introduced data-driven methods that learn the low-dimensional compact representation of designs from data, and presented efficient approaches to explore this compact space.

To understand those data-driven methods, Chapter 2 reviewed some fundamental concepts and the existing state of the art in data-driven design space exploration and synthesis. It first introduced existing design representations and their issues. Then it reviewed the manifold hypothesis and how to learn and explore latent spaces on those manifolds. That chapter also discussed current approaches for synthesizing new designs, namely rule-based, assembly-based, and data-driven approaches.

The dissertation continued by showing how to measure the intrinsic complexity and dimensionality of a design space. The method introduced in Chapter 3 first captures the inherent properties of a design space and then chooses the appropriate design embedding based on the captured properties. It successfully identified multiple sub-manifolds and their intrinsic dimensions. By generating fundamental knowledge about the inherent complexity of a design space and how designs differ from one another, this approach allows us to improve design optimization, consumer preference learning, geometric modeling, and other design applications that rely on navigating complex design spaces. Ultimately, this deepens our understanding of design complexity in general.

The learned properties of the design space can guide data-driven design synthesis. However, purely data-driven methods ignore designs' functional or geometrical constraints and characteristics, and thus will have limitations. Chapters 4 and 5 looked at incorporating prior knowledge into data-driven models to improve synthesis quality. Specifically, Chapter 4 introduced a deep generative model that can synthesize aerodynamic shapes by incorporating smoothness constraints. It also demonstrated that the generative model both (1) learns realistic and compact airfoil shape representations and (2) empirically accelerates optimization convergence by over an order of magnitude. Chapter 5 presented a generative model for synthesizing designs with multiple parts. The method decomposes the problem of synthesizing/optimizing the whole design into synthesizing/optimizing each part separately but keeping the inter-part dependencies satisfied. This technique of capturing dependencies among parts lay the foundation for learned generative models to extend to more realistic engineering systems where such relationships are widespread.

The latent spaces constructed by those above-mentioned models reduce the cost for design space exploration. But sometimes the desired design is remarkably different from existing ones (*i.e.*, outside the region of existing designs in the latent space), which makes it difficult to specify latent space bounds for exploration. Chapters 6 and 7 introduced design space exploration methods that gradually expand a design space or a latent space, so that no fixed bounds are required. Specifically, Chapter 6 introduced a method, Active Expansion Sampling (AES), that identifies feasible domains when the design space/latent space bounds are unclear. The method both learns the domain boundary of feasible designs, while also expanding our knowledge of the design space/latent space as available budget increases. That chapter also showed how coupling design manifolds with AES allows us to actively expand high-dimensional design spaces without incurring this exponential penalty, and it enables

the discovery of designs that have different appearance and functionality from its initial design set. Chapter 7 presented a global optimization method, Trust Region Bayesian Optimization (TRBO), that can be used in a similar scenario, *i.e.*, it only needs to specify an initial search space that does not necessarily include the global optimum, and will expand the search space when necessary. TRBO avoids the over-exploration problem that often occurs in unbounded Bayesian optimization. Results show that it out-performs the current state-of-the-art methods.

8.2 Broader Implications

This dissertation brought together some important aspects for understanding design spaces. It started by looking at the complexity of the design space—the intrinsic dimensionality, the data separability, and the non-linearity. It also showed that we can understand the shape variation in a design space by visualizing a low-dimensional latent space that encodes major shape changes. The dissertation then introduced a way of decomposing a space of assemblies (*i.e.*, designs with multiple components) into hierarchical latent spaces. This not only suits some practical needs (*e.g.*, hierarchical design synthesis/optimization), but also converts a composite design space into subspaces that are more comprehensible by human beings. Through these techniques, we can see clearly what factors change or constrain our designs, so that we can explore alternative solutions within appropriate limits. The techniques of understanding the design space can be extended to any engineering systems where design space exploration is needed and data on previous designs are sufficient. This allows us to explore the design space more efficiently compared to conventional exploration techniques and mitigates the curse of dimensionality.

The unbounded design space exploration methods introduced in Chapters 6 and 7

(*i.e.*, AES and TRBO) essentially eliminate the need for performing adaptive sampling or Bayesian optimization within a fixed bounding box. While this dissertation mainly demonstrated their application in expanding the design space, they can be applied in any circumstances where the variable bounds are unclear. For example, they are useful in deciding the parameters of any engineering systems or algorithms where those parameter bounds are hard to specify, such that any conventional methods that work within fixed bounds would possibly miss either part of the feasible domain (when performing feasible domain identification) or the true global optimum (when performing global optimization) as they may be outside the specified bounds.

8.3 Limitations and Future Research Directions

The main idea of this dissertation is to learn a compact representation of the design space from data, so that it enables efficient design synthesis and design space exploration. However, unlike learning from art or literature (*e.g.*, images, videos, or articles), engineering design usually has strict geometry, functionality, or performance requirements. Violating those requirements may result in invalid designs. This makes purely data-driven models unreliable in practice. Chapters 4 and 6 have demonstrated the combination of a data-driven model with aerodynamic simulation or human annotation, which are used to evaluate or justify synthesized designs. Chapter 4 and 5 also used geometric constraints or inter-part dependencies as prior knowledge to define the data-driven model, so that synthesized designs automatically satisfy those prior beliefs. It is also interesting to incorporate physics into the model so that the synthesized designs will have desired functionality or performance [32, 238]. In the future, we can consider a broader range of prior knowledge regarding the geometry, physical, and mathematical properties of designs.

Designs generated from purely data-driven models will not look much different from existing ones, as these models only learn from past examples. This (potentially) limits the creation of innovative designs. The unbounded design space exploration described in Chapters 6 and 7 mitigated this problem by extending the design manifold. But still, new designs will (by construction) be restricted to the learned manifold. In contrast, techniques like topology optimization, though having its own disadvantages, can generate innovative designs by considering primarily physics-based objectives and constraints. Thus it is interesting to combine machine learning with physics-based solvers to learn a compact solution space for design optimization problems with physics objectives/constraints.

Most design examples used in this work have relatively simple geometries. They are good for demonstrating the concepts and effectiveness of our proposed methods. However, we acknowledge that there are more complex real-world designs and the scalability of our model on data complexity needs to be further investigated in the future. It is also useful to develop techniques for reducing the computational cost for complex designs.

Different from the datasets of images or text that are widely used in the machine learning community, the public access of huge engineering design dataset is unrealistic, which impedes the development of effective data-driven design models. One promising future research avenue would then be improving the sample efficiency of those models. Machine learning researchers have already applied *few-shot learning* to tasks such as classification, regression, and reinforcement learning [72, 200, 222], where the model can learn from only a few examples. Realizing few-shot learning for engineering design tasks will be extremely useful considering the limited design data.

On the other hand, as humans create more and more designs, in the future we are able to maintain a diverse range of design datasets. By leveraging such resource of

existing designs, we can extract useful knowledge for creating new designs and eventually eliminate or reduce the cost of human intervention in the design process. This work has brought us one step closer to the goal where designers can use automated algorithms to efficiently search for any desired solutions, so that it frees the time of designers, provides inspiration for them, and even allows practitioners to create products while minimizing required expert input. This, in turn, will help augment the design datasets and further improve the quality of generated designs.

Appendix A: Publications

A.1 Journal

1. **Chen W**, Fuge M. Synthesizing Designs with Inter-Part Dependencies using Hierarchical Generative Adversarial Networks. ASME. J. Mech. Des. 2019;():1-15. doi:10.1115/1.4044076. (Accepted)

   Code: https://github.com/IDEALLab/hgan_jmd_2019

2. **Chen W**, Fuge M. Active expansion sampling for learning feasible domains in an unbounded input space. Structural and Multidisciplinary Optimization. 2018;57(3):925-945. doi:10.1007/s00158-017-1894-y.

   Code: https://github.com/IDEALLab/Active-Expansion-Sampling

3. **Chen W**, Fuge M. Beyond the Known: Detecting Novel Feasible Domains over an Unbounded Design Space. ASME. J. Mech. Des. 2017;139(11):111405-111405-10. doi:10.1115/1.4037306.

   Code: https://github.com/IDEALLab/domain_expansion_jmd_2017

4. **Chen W**, Fuge M, Chazan J. Design Manifolds Capture the Intrinsic Complexity and Dimension of Design Spaces. ASME. J. Mech. Des. 2017;139(5):051102-051102-10. doi:10.1115/1.4036134.

   Code: https://github.com/IDEALLab/design_embeddings_jmd_2016

A.2 Peer-Reviewed Conference

1. **Chen W**, Fuge M. Expanding Search Space for Global Optimization with Trust Region Bayesian Optimization. Advances in Neural Information Processing Systems. 2019. (under review)

2. **Chen W**, Chiu K, Fuge M. Aerodynamic Design Optimization and Shape Exploration using Generative Adversarial Networks. AIAA Scitech 2019 Forum. doi:10.2514/6.2019-2351. (Invited talk)

   Code: https://github.com/IDEALLab/airfoil-opt-gan

3. **Chen W**, Jeyaseelan A, Fuge M. Synthesizing Designs With Inter-Part Dependencies Using Hierarchical Generative Adversarial Networks. ASME. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 2A: 44th Design Automation Conference:V02AT03A007. doi:10.1115/DETC2018-85339.

   Code: https://github.com/IDEALLab/hgan_idetc2018

4. **Chen W**, Chazan J, Fuge M. How Designs Differ: Non-Linear Embeddings Illuminate Intrinsic Design Complexity. ASME. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 2A: 42nd Design Automation Conference:V02AT03A014. doi:10.1115/DETC2016-60112.

   Code: https://github.com/IDEALLab/design_embeddings_idetc_2016

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alekh Agarwal. Selective sampling algorithms for cost-sensitive multiclass prediction. *ICML (3)*, 28:1220–1228, 2013.

[3] Ibrahim Alabdulmohsin, Xin Gao, and Xiangliang Zhang. Efficient active learning of halfspaces via query synthesis. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2483–2489. AAAI Press, 2015.

[4] Anand Amrit, Leifur T Leifsson, Slawomir Koziel, and Yonatan Afework Tesfahunegn. Efficient multi-objective aerodynamic optimization by design space dimension reduction and co-kriging. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 3515, 2016.

[5] George R Anderson, Marian Nemec, and Michael J Aftosmis. Aerodynamic shape optimization benchmarks with error control and automatic parameterization. In *53rd AIAA Aerospace Sciences Meeting*, page 1719, 2015.

[6] Dana Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

[7] Shlomo Argamon-Engelson and Ido Dagan. Committee-based sample selection for probabilistic classifiers. *J. Artif. Intell. Res.(JAIR)*, 11:335–360, 1999.

[8] Melinos Averkiou, Vladimir G Kim, Youyi Zheng, and Niloy J Mitra. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In *Computer Graphics Forum*, volume 33, pages 125–134. Wiley Online Library, 2014.

[9] Pranjal Awasthi, Vitaly Feldman, and Varun Kanade. Learning using local membership queries. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 398–431, Princeton, NJ, USA, 12–14 Jun 2013. PMLR.

[10] Yoram Baram, Ran El Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5(Mar):255–291, 2004.

[11] Anirban Basudhar, Christoph Dribusch, Sylvain Lacaze, and Samy Missoum. Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization*, 46(2):201–221, 2012.

[12] Anirban Basudhar and Samy Missoum. Adaptive explicit decision functions for probabilistic design and optimization using support vector machines. *Computers & Structures*, 86(19):1904–1917, 2008.

[13] Anirban Basudhar and Samy Missoum. An improved adaptive sampling scheme for the construction of explicit boundaries. *Structural and Multidisciplinary Optimization*, 42(4):517–529, 2010.

[14] R.E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NY, 1957.

[15] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[16] Martin P Bendsøe. *Topology optimization*. Springer, 2009.

[17] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, Aug 2013.

[18] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[19] Steven H Berguin and Dimitri N Mavris. Dimensional design space exploration of expensive functions with access to gradient. In *15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 2174, 2014.

[20] Steven H Berguin and Dimitri N Mavris. Dimensionality reduction using principal component analysis applied to the gradient. *AIAA Journal*, 53(4):1078–1090, 2014.

[21] Steven H Berguin, David Rancourt, and Dimitri N Mavris. Method to facilitate high-dimensional design space exploration using computationally expensive analyses. *AIAA Journal*, 53(12):3752–3765, 2015.

[22] Djallel Bouneffouf. Exponentiated gradient exploration for active learning. *Computers*, 5(1):1, 2016.

[23] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[24] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

[25] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[26] Brent Bryan, Robert C Nichol, Christopher R Genovese, Jeff Schneider, Christopher J Miller, and Larry Wasserman. Active learning for identifying function threshold boundaries. In *Advances in neural information processing systems*, pages 163–170, 2006.

[27] Alexander Burnap, Ye Liu, Yanxin Pan, Honglak Lee, Richard Gonzalez, and Panos Y Papalambros. Estimating and exploring the product form design space using deep generative models. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V02AT03A013–V02AT03A013. American Society of Mechanical Engineers, 2016.

[28] Alexander Burnap, Ye Liu, Yanxin Pan, Honglak Lee, Richard Gonzalez, and Panos Y Papalambros. Estimating and exploring the product form design space using deep generative models. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, page V02AT03A013. American Society of Mechanical Engineers, 2016.

[29] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[30] Jonathan Cagan, Matthew I Campbell, Susan Finger, and Tetsuo Tomiyama. A framework for computational design synthesis: model and applications. *Journal of Computing and Information Science in Engineering*, 5(3):171–181, 2005.

[31] Colin Campbell, Nello Cristianini, and Alex J Smola. Query learning with large margin classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 111–118. Morgan Kaufmann Publishers Inc., 2000.

[32] Ruijin Cang, Hope Yao, and Yi Ren. One-shot generation of near-optimal topology through theory-driven machine learning. *Computer-Aided Design*, 109:12–21, 2019.

[33] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Linear classification and selective sampling under low noise conditions. In *Advances in Neural Information Processing Systems*, pages 249–256, 2009.

[34] Nicolo Cesa-Bianchi, Claudio Gentile, and Francesco Orabona. Robust bounds for classification via selective sampling. In *Proceedings of the 26th annual international conference on machine learning*, pages 121–128. ACM, 2009.

[35] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. Attribit: content creation with semantic attributes. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 193–202. ACM, 2013.

[36] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. In *ACM Transactions on Graphics (TOG)*, volume 30, page 35. ACM, 2011.

[37] Siddhartha Chaudhuri and Vladlen Koltun. Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics (TOG)*, 29(6):183, 2010.

[38] Lin Chen, Hamed Hassani, and Amin Karbasi. Near-optimal active learning of halfspaces via query synthesis in the noisy setting. *arXiv preprint arXiv:1603.03515*, 2016.

[39] Tian Qi Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. *arXiv preprint arXiv:1802.04942*, 2018.

[40] Wei Chen, Jonah Chazan, and Mark Fuge. How designs differ: Non-linear embeddings illuminate intrinsic design complexity. In *ASME International Design Engineering Technical Conferences*. ASME, August 2016.

[41] Wei Chen, Noa Chazan, and Mark Fuge. How designs differ: Non-linear embeddings illuminate intrinsic design complexity. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V02AT03A014–V02AT03A014. American Society of Mechanical Engineers, August 2016.

[42] Wei Chen, Kevin Chiu, and Mark Fuge. Aerodynamic design optimization and shape exploration using generative adversarial networks. In *AIAA Scitech 2019 Forum*, page 2351, 2019.

[43] Wei Chen and Mark Fuge. Active expansion sampling for learning feasible domains in an unbounded input space. *Structural and Multidisciplinary Optimization*, pages 1–21, 2017.

[44] Wei Chen and Mark Fuge. Beyond the known: Detecting novel feasible domains over an unbounded design space. *Journal of Mechanical Design*, 139(11):111405–111405–10, 2017.

[45] Wei Chen and Mark Fuge. Active expansion sampling for learning feasible domains in an unbounded input space. *Structural and Multidisciplinary Optimization*, 57(3):925–945, 2018.

[46] Wei Chen and Mark Fuge. Béziergan: Automatic generation of smooth curves from interpretable low-dimensional parameters. *arXiv preprint arXiv:1808.08871*, 2018.

[47] Wei Chen and Mark Fuge. Béziergan: Automatic generation of smooth curves from interpretable low-dimensional parameters. *arXiv preprint arXiv:1808.08871*, 2018.

[48] Wei Chen and Mark Fuge. Synthesizing designs with inter-part dependencies using hierarchical generative adversarial networks. *Journal of Mechanical Design*, 2019.

[49] Wei Chen, Mark Fuge, and Noa Chazan. Design manifolds capture the intrinsic complexity and dimension of design spaces. *Journal of Mechanical Design*, 139(5):051102–051102–10, 2017.

[50] Wei Chen, Ashwin Jeyaseelan, and Mark Fuge. Synthesizing designs with inter-part dependencies using hierarchical generative adversarial networks. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Quebec City, Canada, Aug 2018. ASME.

[51] Xi Chen, Matteo Diez, Manivannan Kandasamy, Zhiguo Zhang, Emilio F Campana, and Frederick Stern. High-fidelity global optimization of shape design by dimensionality reduction, metamodels and deterministic particle swarm. *Engineering Optimization*, 47(4):473–494, 2015.

[52] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[53] Zhenzhong Chen, Siping Peng, Xiaoke Li, Haobo Qiu, Huadi Xiong, Liang Gao, and Peigen Li. An important boundary sampling method for reliability-based design optimization using kriging model. *Structural and Multidisciplinary Optimization*, 52(1):55–70, 2015.

[54] Zhenzhong Chen, Haobo Qiu, Liang Gao, Xiaoke Li, and Peigen Li. A local adaptive sampling method for reliability-based design optimization using kriging model. *Structural and Multidisciplinary Optimization*, 49(3):401–416, 2014.

[55] Davide Cinquegrana and Emiliano Iuliano. Efficient global optimization of a transonic wing with geometric data reduction. In *35th AIAA Applied Aerodynamics Conference*, page 3057, 2017.

[56] Davide Cinquegrana and Emiliano Iuliano. Investigation of adaptive design variables bounds in dimensionality reduction for aerodynamic shape optimization. *Computers & Fluids*, 174:89–109, 2018.

[57] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

[58] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *In Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[59] Danny D'Agostino, Andrea Serani, Emilio F Campana, and Matteo Diez. Deep autoencoder for off-line design-space dimensionality reduction in shape optimization. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 1648, 2018.

[60] Sanjoy Dasgupta, Adam Tauman Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. *Journal of Machine Learning Research*, 10(Feb):281–299, 2009.

[61] Ofer Dekel, Claudio Gentile, and Karthik Sridharan. Selective sampling and active learning from single and multiple teachers. *Journal of Machine Learning Research*, 13(Sep):2655–2697, 2012.

[62] Pierluigi Della Vecchia, Elia Daniele, and Egidio D'Amato. An airfoil shape optimization technique coupling parsec parameterization and evolutionary algorithm. *Aerospace Science and Technology*, 32(1):103–110, 2014.

[63] RW Derksen and Tim Rogalsky. Bezier-parsec: An optimized aerofoil parameterization for design. *Advances in engineering software*, 41(7-8):923–930, 2010.

[64] Srikanth Devanathan and Karthik Ramani. Creating polytope representations of design spaces for visual exploration using consistency techniques. *Journal of Mechanical Design*, 132(8):081011, 2010.

[65] Tristan Dhert, Turaj Ashuri, and Joaquim R. R. A. Martins. Aerodynamic shape optimization of wind turbine blades using a Reynolds-averaged Navier–Stokes model and an adjoint method. *Wind Energy*, 20:909–926, 05/2017 2017.

[66] Matteo Diez, Emilio F Campana, and Frederick Stern. Design-space dimensionality reduction in shape optimization by karhunen–loève expansion. *Computer Methods in Applied Mechanics and Engineering*, 283:1525–1544, 2015.

[67] Mark Drela. Xfoil: An analysis and design system for low reynolds number airfoils. In *Low Reynolds number aerodynamics*, pages 1–12. Springer, 1989.

[68] David K Duvenaud, Oren Rippel, Ryan P Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *AISTATS*, pages 202–210, 2014.

[69] Danny DAgostino, Andrea Serani, Emilio F Campana, and Matteo Diez. Nonlinear methods for design-space dimensionality reduction in shape optimization. In *International Workshop on Machine Learning, Optimization, and Big Data*, pages 121–132. Springer, 2017.

[70] M Ebrahimi and A Jahangirian. Aerodynamic optimization of airfoils using adaptive parameterization and genetic algorithm. *Journal of Optimization Theory and Applications*, 162(1):257–271, 2014.

[71] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 38, page 1, 2017.

[72] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[73] Noa Fish, Melinos Averkiou, Oliver Van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J Mitra. Meta-representation of shape families. *ACM Transactions on Graphics (TOG)*, 33(4):34, 2014.

[74] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[75] Yoav Freund, H Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine learning*, 28(2):133–168, 1997.

[76] Francesca Fusi, Giuseppe Quaranta, Alberto Guardone, and Pietro M Congedo. Drag minimization of an isolated airfoil in transonic inviscid flow by means of genetic algorithms. In *53rd AIAA Aerospace Sciences Meeting*, page 1722, 2015.

[77] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Aerodynamic design exploration through surrogate-assisted illumination. In *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 3330, 2017.

[78] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 250–259. AUAI Press, 2014.

[79] Johan Gielis. A generic geometric transformation that unifies a wide range of natural and abstract shapes. *American journal of botany*, 90(3):333–338, 2003.

[80] Thomas Gmeiner and Kristina Shea. A spatial grammar for the computational design synthesis of vise jaws. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V03AT03A006–V03AT03A006. American Society of Mechanical Engineers, 2013.

[81] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[82] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[83] Dian Gong, Xuemei Zhao, and Gérard Medioni. Robust multiple manifolds structure learning. *arXiv preprint arXiv:1206.4624*, 2012.

[84] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[85] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[86] Alkis Gotovos, Nathalie Casati, Gregory Hitz, and Andreas Krause. Active learning for level set estimation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1344–1350. AAAI Press, 2013.

[87] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[88] Zachary J Grey and Paul G Constantine. Active subspaces of airfoil shape parameterizations. *AIAA Journal*, 56(5):2003–2017, 2018.

[89] Tinghao Guo, Daniel R Herber, and James T Allison. Circuit synthesis using generative adversarial networks (GANs). In *AIAA 2019 Science and Technology Forum and Exposition*, number AIAA 2019-2350, San Diego, CA, USA, January 2019.

[90] Xuekun Guo, Juncong Lin, Kai Xu, and Xiaogang Jin. Creature grammar for creative modeling of 3d monsters. *Graphical Models*, 76(5):376–389, 2014.

[91] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

[92] Zhong-Hua Han, Jing Chen, Ke-Shi Zhang, Zhen-Ming Xu, Zhen Zhu, and Wen-Ping Song. Aerodynamic shape optimization of natural-laminar-flow wing using surrogate-based approach. *AIAA Journal*, 56(7):2579–2593, 2018.

[93] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

[94] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

[95] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[96] Trong Nghia Hoang, Bryan Kian Hsiang Low, Patrick Jaillet, and Mohan Kankanhalli. Nonmyopic ϵ-bayes-optimal active learning of gaussian processes. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 739–747, Bejing, China, 22–24 Jun 2014. PMLR.

[97] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Semisupervised svm batch mode active learning with applications to image retrieval. *ACM Transactions on Information Systems (TOIS)*, 27(3):16, 2009.

[98] Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2659–2665. AAAI Press, 2015.

[99] Haibin Huang, Evangelos Kalogerakis, and Benjamin Marlin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. In *Computer Graphics Forum*, volume 34, pages 25–38. Wiley Online Library, 2015.

[100] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. In *Advances in neural information processing systems*, pages 892–900, 2010.

[101] Yen-Chih Huang and Kuei-Yuan Chan. A modified efficient global optimization algorithm for maximal reliability in a probabilistic constrained space. *Journal of Mechanical Design*, 132(6):061002, 2010.

[102] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[103] Jeffrey C Jackson. An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997.

[104] A. R. Jahangirian and M. Ebrahimi. Airfoil shape optimization with adaptive mutation genetic algorithm. *Journal of Aerospace Science and Technology*, 11(1):–, 2017.

[105] Jé, r iuml, me Lé, pine, Fran-atilde, ois Guibault, Jean-Yves Tré, panier, Fran-atilde, ois Pé, et al. Optimized nonuniform rational b-spline geometrical representation for aerodynamic design of wings. *AIAA journal*, 39(11):2033–2041, 2001.

[106] Jae-Ho Jeong and Soo-Hyun Kim. Optimization of thick wind turbine airfoils using a genetic algorithm. *Journal of Mechanical Science and Technology*, 32(7):3191–3199, Jul 2018.

[107] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[108] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[109] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG)*, 31(4):55, 2012.

[110] Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. Query efficient posterior estimation in scientific experiments via bayesian active learning. *Artificial Intelligence*, 243:45–56, 2017.

[111] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Gaussian processes for object categorization. *International journal of computer vision*, 88(2):169–188, 2010.

[112] Gaetan K. W. Kenway, Graeme J. Kennedy, and Joaquim R. R. A. Martins. Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations. *AIAA Journal*, 52:935–951, 2014.

[113] Gaetan KW Kenway and Joaquim RRA Martins. Buffet-onset constraint formulation for aerodynamic shape optimization. *AIAA Journal*, 55(6):1930–1947, 2017.

[114] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[115] Ross D King, Kenneth E Whelan, Ffion M Jones, Philip GK Reiser, Christopher H Bryant, Stephen H Muggleton, Douglas B Kell, and Stephen G Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, 2004.

[116] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[117] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[118] Corinna Königseder and Kristina Shea. Analyzing generative design grammars. In *Design Computing and Cognition'14*, pages 363–381. Springer International Publishing, 2015.

[119] Corinna Königseder and Kristina Shea. Comparing strategies for topologic and parametric rule application in automated computational design synthesis. *Journal of Mechanical Design*, 138(1):011102, 2016.

[120] Corinna Königseder and Kristina Shea. Visualizing relations between grammar rules, objectives, and search space exploration in grammar-based computational design synthesis. *Journal of Mechanical Design*, 138(10):101101, 2016.

[121] Corinna Königseder, Tino Stanković, and Kristina Shea. Improving design grammar development and application through network-based analysis of transition graphs. *Design Science*, 2, 2016.

[122] Konstantinos V Kostas, Alexandros I Ginnis, Constantinos G Politis, and Panagiotis D Kaklis. Shape-optimization of 2d hydrofoils using an isogeometric bem solver. *Computer-Aided Design*, 82:79–87, 2017.

[123] Slawomir Koziel and Leifur Leifsson. Multi-level surrogate-based airfoil shape optimization. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 778, 2013.

[124] Andreas Krause and Carlos Guestrin. Nonmyopic active learning of gaussian processes: an exploration-exploitation approach. In *Proceedings of the 24th international conference on Machine learning*, pages 449–456. ACM, 2007.

[125] Georg Krempl, Daniel Kottke, and Vincent Lemaire. Optimised probabilistic active learning (opal). *Machine Learning*, 100(2-3):449–476, 2015.

[126] Brenda M Kulfan. Universal parametric geometry representation method. *Journal of Aircraft*, 45(1):142–158, 2008.

[127] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[128] Brad J Larson and Christopher A Mattson. Design space exploration for quantifying a system models feasible domain. *Journal of Mechanical Design*, 134(4):041010, 2012.

[129] Johan Larsson and Qiqi Wang. The prospect of using large eddy and detached eddy simulations in engineering design, and the research required to get there. *Phil. Trans. R. Soc. A*, 372(2022):20130329, 2014.

[130] Guenhael Le Quilliec, Balaji Raghavan, and Piotr Breitkopf. A manifold learning-based reduced order model for springback shape characterization and optimization in sheet metal forming. *Computer Methods in Applied Mechanics and Engineering*, 285:621–638, 2015.

[131] Rodolphe Le Riche and Raphael T Haftka. Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm. *AIAA journal*, 31(5):951–956, 1993.

[132] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer, New York, 2007.

[133] Tae Hee Lee and Jae Jun Jung. A sampling technique enhancing accuracy and efficiency of metamodel-based rbdo: Constraint boundary sampling. *Computers & Structures*, 86(13):1463–1476, 2008.

[134] Jerome Lepine, Jean-Yves Trepanier, and Francois Pepin. Wing aerodynamic design using an optimized nurbs geometrical representation. In *38th Aerospace Sciences Meeting and Exhibit*, page 669, 2000.

[135] David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, pages 148–156, 1994.

[136] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[137] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):52, 2017.

[138] Pei Li, Richard Seebass, and Helmut Sobieczky. Manual aerodynamic optimization of an oblique flying wing. In *36th AIAA Aerospace Sciences Meeting and Exhibit*, page 598, 1998.

[139] Zijing LIU, Xuejun LIU, and Xinye CAI. A new hybrid aerodynamic optimization framework based on differential evolution and invasive weed optimization. *Chinese Journal of Aeronautics*, 31(7):1437 – 1448, 2018.

[140] Danny J Lohan, Ercan M Dede, and James T Allison. Topology optimization for heat conduction using generative design algorithms. *Structural and Multidisciplinary Optimization*, 55(3):1063–1077, 2017.

[141] Trent W Lukaczyk, Paul Constantine, Francisco Palacios, and Juan J Alonso. Active subspaces for shape optimization. In *10th AIAA Multidisciplinary Design Optimization Conference*, page 1171, 2014.

[142] Yifei Ma, Roman Garnett, and Jeff Schneider. Active area search via bayesian quadrature. In *Artificial Intelligence and Statistics*, pages 595–603, 2014.

[143] Oisin Mac Aodha, Neill DF Campbell, Jan Kautz, and Gabriel J Brostow. Hierarchical subquery evaluation for active learning on a graph. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2014.

[144] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529, 1993.

[145] K Mani, B.A. Lockwood, and D.J. Mavriplis. Adjoint-based unsteady airfoil design optimization with application to dynamic stall. *Annual Forum Proceedings - AHS International*, 3:1940–1952, 01 2012.

[146] Dominic A Masters, Nigel J Taylor, TCS Rendall, Christian B Allen, and Daniel J Poole. Geometric comparison of aerofoil shape parameterization methods. *AIAA Journal*, pages 1575–1589, 2017.

[147] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. Dream lens: Exploration and visualization of large-scale generative design datasets. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 369. ACM, 2018.

[148] Andrew McCallum, Kamal Nigam, et al. Employing em and pool-based active learning for text classification. In *ICML*, volume 98, pages 359–367, 1998.

[149] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[150] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *Acm Transactions On Graphics (Tog)*, volume 25, pages 614–623. ACM, 2006.

[151] Raymond H Myers, Douglas C Montgomery, et al. *Response surface methodology: process and product optimization using designed experiments*, volume 3. Wiley New York, 1995.

[152] Siva Nadarajah, Patrice Castonguay, and Arash Mousavi. Survey of shape parameterization techniques and its effect on three-dimensional aerodynamic shape optimization. In *18th AIAA computational fluid dynamics conference*, page 3837, 2007.

[153] Charlie Nash and Chris KI Williams. The shape variational autoencoder: A deep generative model of part-segmented 3d objects. In *Computer Graphics Forum*, volume 36, pages 1–12. Wiley Online Library, 2017.

[154] Hieu T. Nguyen and Arnold Smeulders. Active learning using pre-clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 79–, New York, NY, USA, 2004. ACM.

[155] Vu Nguyen, Sunil Gupta, Santu Rana, Cheng Li, and Svetha Venkatesh. Filtering bayesian optimization approach in weakly specified search space. *Knowledge and Information Systems*, pages 1–29, 2018.

[156] Vu Nguyen, Sunil Gupta, Santu Rane, Cheng Li, and Svetha Venkatesh. Bayesian optimization in weakly specified search space. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 347–356. IEEE, 2017.

[157] Horst Nowacki. Modelling of design decisions for cad. In *Computer Aided Design Modelling, Systems Engineering, CAD-Systems*, pages 177–223. Springer, 1980.

[158] Matthias Oberhauser, Sky Sartorius, Thomas Gmeiner, and Kristina Shea. Computational design synthesis of aircraft configurations with shape grammars. In *Design Computing and Cognition'14*, pages 21–39. Springer International Publishing, 2015.

[159] Katsunori Ohnishi, Shohei Yamamoto, Yoshitaka Ushiku, and Tatsuya Harada. Hierarchical video generation from orthogonal information: Optical flow and texture. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[160] Francesco Orabona and Nicolo Cesa-Bianchi. Better algorithms for selective sampling. In *Proceedings of the 28th international conference on Machine learning (ICML-11)*, pages 433–440, 2011.

[161] Thomas Osugi, Deng Kim, and Stephen Scott. Balancing exploration and exploitation: A new algorithm for active machine learning. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005.

[162] Maks Ovsjanikov, Wilmot Li, Leonidas Guibas, and Niloy J Mitra. Exploration of continuous variability in collections of 3d shapes. In *ACM Transactions on Graphics (TOG)*, volume 30, page 33. ACM, 2011.

[163] Simon Painchaud-Ouellet, Christophe Tribes, Jean-Yves Trépanier, and Dominique Pelletier. Airfoil shape optimization using a nonuniform rational b-splines parametrization under thickness constraint. *AIAA journal*, 44(10):2170–2178, 2006.

[164] Pramudita Satria Palar and Koji Shimoyama. Efficient global optimization with ensemble and selection of kernel functions for engineering design. *Structural and Multidisciplinary Optimization*, 59(1):93–116, 2019.

[165] Haobo Qiu, Yanjiao Xu, Liang Gao, Xiaoke Li, and Li Chi. Multi-stage design space reduction and metamodeling optimization method based on self-organizing maps and fuzzy clustering. *Expert Systems with Applications*, 46:180–195, 2016.

[166] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[167] Balaji Raghavan, Piotr Breitkopf, Yves Tourbier, and Pierre Villon. Towards a space reduction approach for efficient structural shape optimization. *Structural and Multidisciplinary Optimization*, 48(5):987–1000, 2013.

[168] Balaji Raghavan, Guenhael Le Quilliec, Piotr Breitkopf, Alain Rassineux, Jean-Marc Roelandt, and Pierre Villon. Numerical assessment of springback for the deep drawing process by level set interpolation using shape manifolds. *International Journal of Material Forming*, 7(4):487–501, 2014.

[169] Balaji Raghavan, Liang Xia, Piotr Breitkopf, Alain Rassineux, and Pierre Villon. Towards simultaneous reduction of both input and output spaces for interactive simulation-based structural design. *Computer Methods in Applied Mechanics and Engineering*, 265:174–185, 2013.

[170] Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. In *Advances in Neural Information Processing Systems*, pages 280–288, 2016.

[171] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[172] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. the MIT Press, 2006.

[173] T Ray and HM Tsai. Swarm algorithm for single-and multiobjective airfoil design optimization. *AIAA journal*, 42(2):366–373, 2004.

[174] Jeffrey C Regier and Philip B Stark. Mini-minimax uncertainty quantification for emulators. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):686–708, 2015.

[175] Yi Ren and Panos Y Papalambros. A design preference elicitation query as an optimization process. *Journal of Mechanical Design*, 133(11):111004, 2011.

[176] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 3, 2017.

[177] Filipe Rodrigues, Francisco Pereira, and Bernardete Ribeiro. Gaussian process classification and active learning with multiple annotators. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 433–441. JMLR Workshop and Conference Proceedings, 2014.

[178] T Rogalsky, S Kocabiyik, and RW Derksen. Differential evolution in aerodynamic optimization. *Canadian Aeronautics and Space Journal*, 46(4):183–190, 2000.

[179] Jamshid A Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA journal*, 39(5):877–884, 2001.

[180] Oksana Samko, A David Marshall, and Paul L Rosin. Selection of the optimal parameter value for the isomap algorithm. *Pattern Recognition Letters*, 27(9):968–979, 2006.

[181] Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *ICML*, pages 839–846, 2000.

[182] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[183] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[184] Matthias Schramm, Bernhard Stoevesandt, and Joachim Peinke. Simulation and optimization of an airfoil with leading edge slat. *Journal of Physics: Conference Series*, 753(2):022052, 2016.

[185] Matthias Schramm, Bernhard Stoevesandt, and Joachim Peinke. Optimization of airfoils using the adjoint approach and the influence of adjoint turbulent viscosity. *Computation*, 6(1):5, Jan 2018.

[186] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*, 20(4):151–160, 1986.

[187] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[188] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics, 2008.

[189] Bobak Shahriari. *Practical Bayesian optimization with application to tuning machine learning algorithms*. PhD thesis, University of British Columbia, 2016.

[190] Bobak Shahriari, Alexandre Bouchard-Côté, and Nando Freitas. Unbounded bayesian optimization via regularization. In *Artificial Intelligence and Statistics*, pages 1168–1176, 2016.

[191] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[192] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[193] Viktoriia Sharmanska, Daniel Hernández-Lobato, Jose Miguel Hernandez-Lobato, and Novi Quadrianto. Ambiguity helps: Classification with disagreements in crowdsourced annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2194–2202, 2016.

[194] Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Trans. Graph.*, 30(6):126:1–126:10, December 2011.

[195] Prashant Singh, Joachim Van Der Herten, Dirk Deschrijver, Ivo Couckuyt, and Tom Dhaene. A sequential sampling strategy for adaptive classification of computationally expensive data. *Structural and Multidisciplinary Optimization*, 55(4):1425–1438, 2017.

[196] Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3d shape surfaces using geometry images. In *European Conference on Computer Vision*, pages 223–240. Springer, 2016.

[197] Ayan Sinha, Asim Unmesh, Qixing Huang, and Karthik Ramani. Surfnet: Generating 3d shape surfaces using deep residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6040–6049, 2017.

[198] Shaun N Skinner and Hossein Zare-Behtash. State-of-the-art in aerodynamic shape optimisation methods. *Applied Soft Computing*, 62:933–962, 2018.

[199] Edward Smith and David Meger. Improved adversarial systems for 3d object generation and reconstruction. *arXiv preprint arXiv:1707.09557*, 2017.

[200] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[201] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[202] Helmut Sobieczky. Parametric airfoils and wings. In *Recent development of aerodynamic design methodologies*, pages 71–87. Springer, 1999.

[203] Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D Kulkarni, and Joshua B Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *The IEEE conference on computer vision and pattern recognition (CVPR)*, volume 3, page 4, 2017.

[204] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142, 2016.

[205] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[206] Niranjan Srinivas, Andreas Krause, Matthias Seeger, and Sham M Kakade. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1015–1022, 2010.

[207] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013.

[208] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 63–74. ACM, 2012.

[209] Jerry O Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. Exploratory modeling with collaborative design spaces. *ACM Transactions on Graphics-TOG*, 28(5):167, 2009.

[210] Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Transactions on Graphics (TOG)*, 30(2):11, 2011.

[211] Emad Tandis and Ehsanolah Assareh. Inverse design of airfoils via an intelligent hybrid optimization technique. *Engineering with Computers*, 33(3):361–374, Jul 2017.

[212] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, volume 2, page 8, 2017.

[213] Yonatan A. Tesfahunegn, Slawomir Koziel, Leifur Leifsson, and Adrian Bekasiewicz. Surrogate-based airfoil design with space mapping and adjoint sensitivity. *Procedia Computer Science*, 51:795 – 804, 2015. International Conference On Computational Science, ICCS 2015.

[214] Marco Tezzele, Filippo Salmoiraghi, Andrea Mola, and Gianluigi Rozza. Dimension reduction in heterogeneous parametric spaces with application to naval engineering shape design problems. *arXiv preprint arXiv:1709.03298*, 2017.

[215] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[216] LJP Van der Maaten, EO Postma, and HJ Van den Herik. Dimensionality reduction: A comparative review. *Technical Report TiCC TR 2009-005*, 2009.

[217] Emmanuel Vazquez and Julien Bect. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and inference*, 140(11):3088–3095, 2010.

[218] P Venkataraman. A new procedure for airfoil definition. In *13th Applied Aerodynamics Conference*, page 1875, 1995.

[219] Gerhard Venter and Jaroslaw Sobieszczanski-Sobieski. Particle swarm optimization. *AIAA journal*, 41(8):1583–1589, 2003.

[220] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[221] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[222] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[223] Asha Viswanath, AIJ Forrester, and AJ Keane. Constrained design optimization using generative topographic mapping. *AIAA journal*, 52(5):1010–1023, 2014.

[224] Asha Viswanath, AI J. Forrester, and AJ Keane. Dimension reduction for aerodynamic design optimization. *AIAA journal*, 49(6):1256–1266, 2011.

[225] Kun Wang, Shengjiao Yu, and Tiegang Liu. Airfoil optimization based on isogeometric discontinuous galerkin. In *Proceedings of the 2018 2Nd International Conference on Algorithms, Computing and Systems*, ICACS '18, pages 227–231, New York, NY, USA, 2018. ACM.

[226] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017.

[227] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *European Conference on Computer Vision*, pages 318–335. Springer, 2016.

[228] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[229] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3627–3635. JMLR. org, 2017.

[230] William J Welch, Robert J Buck, Jerome Sacks, Henry P Wynn, Toby J Mitchell, and Max D Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[231] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[232] David F Wyatt, David C Wynn, Jerome P Jarrett, and P John Clarkson. Supporting product architecture design using computational design synthesis with network structure constraints. *Research in Engineering Design*, 23(1):17–52, 2012.

[233] Kai Xu, Vladimir G Kim, Qixing Huang, Niloy Mitra, and Evangelos Kalogerakis. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses*, page 4. ACM, 2016.

[234] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Fit and diverse: set evolution for inspiring 3d shape galleries. *ACM Transactions on Graphics (TOG)*, 31(4):57, 2012.

[235] Xufeng Yang, Yongshou Liu, Yi Gao, Yishang Zhang, and Zongzhan Gao. An active learning kriging model for hybrid reliability analysis with both random and interval variables. *Structural and Multidisciplinary Optimization*, 51(5):1003–1016, 2015.

[236] Yi Yang, Zhigang Ma, Feiping Nie, Xiaojun Chang, and Alexander G Hauptmann. Multi-class active learning by uncertainty sampling with diversity maximization. *International Journal of Computer Vision*, 113(2):113–127, 2015.

[237] Bernard Yannou, Faysal Moreno, Henri J Thevenot, and Timothy W Simpson. Faster generation of feasible design points. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 355–363. American Society of Mechanical Engineers, 2005.

[238] Houpu Yao, Yi Ren, and Yongming Liu. Fea-net: A deep convolutional neural network with physicsprior for efficient data driven pde learning. In *AIAA Scitech 2019 Forum*, page 0680, 2019.

[239] QIU Yasong, BAI Junqiang, LIU Nan, and WANG Chen. Global aerodynamic design optimization based on data dimensionality reduction. *Chinese Journal of Aeronautics*, 31(4):643–659, 2018.

[240] Mehmet Ersin Yumer, Paul Asente, Radomir Mech, and Levent Burak Kara. Procedural modeling using autoencoder networks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 109–118, New York, NY, USA, 2015. ACM.

[241] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K. Hodgins, and Levent Burak Kara. Semantic shape editing using deformation handles. *ACM Trans. Graph.*, 34(4):86:1–86:12, July 2015.

[242] Mehmet Ersin Yumer and Levent Burak Kara. Co-constrained handles for deformation in shape collections. *ACM Transactions on Graphics (TOG)*, 33(6):187, 2014.

[243] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[244] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *NIPS*, volume 17, page 16, 2004.

[245] Zhenyue Zhang, Jing Wang, and Hongyuan Zha. Adaptive manifold learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):253–265, 2012.

[246] Youyi Zheng, Daniel Cohen-Or, and Niloy J. Mitra. Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum (Eurographics)*, 32(2pt2):195–204, 2013.

[247] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, volume 3, 2003.

[248] Xiaotian Zhuang and Rong Pan. A sequential sampling strategy to improve reliability-based design optimization with implicit constraint functions. *Journal of Mechanical Design*, 134(2):021002, 2012.

[249] Lavi R Zuhal, Cahya Amalinadhi, Yohanes B Dwianto, Pramudita S Palar, and Koji Shimoyama. Benchmarking multi-objective bayesian global optimization strategies for aerodynamic design. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0914, 2018.