Algorithm Selection in Structural Optimization

by

Rory Clune

B.Eng. (Civil), National University of Ireland, Cork (2008)

S.M., Massachusetts Institute of Technology (2010)

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in the Field of Civil Engineering and Computation
at the
Massachusetts Institute of Technology

June 2013

Signature of Author:

Department of Civil and Environmental Engineering
May 3, 2013

Certified by:

Jerome J. Connor
Professor of Civil and Environmental Engineering
Thesis Co-Supervisor

Certified by:

John A. Ochsendorf
Professor of Building Technology and Civil and Environmental Engineering
Thesis Co-Supervisor

Accepted by:

Heidi M. Nepf
Chair, Departmental Committee for Graduate Students

Algorithm Selection in Structural Optimization

by

Rory Clune

Submitted to the Department of Civil and Environmental Engineering
on May 3, 2013 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy
in the field of Civil Engineering and Computation

Abstract

Structural optimization is largely unused as a practical design tool, despite an extensive academic literature which demonstrates its potential to dramatically improve design processes and outcomes. Many factors inhibit optimization's application. Among them is the requirement for engineers—who generally lack the requisite expertise—to choose an optimization algorithm for a given problem. A suitable choice of algorithm improves the resulting design and reduces computational cost, yet the field of optimization does little to guide engineers in selecting from an overwhelming number of options. The goal of this dissertation is to aid, and ultimately to automate, algorithm selection, thus enhancing optimization's applicability in real-world design.

The initial chapters examine the extent of the problem by reviewing relevant literature and by performing a short, empirical study of algorithm performance variation. We then specify hundreds of bridge design problems by methodically varying problem characteristics, and solve each of them with eight commonly-used nonlinear optimization algorithms. The resulting, extensive data set is used to address the algorithm selection problem.

The results are first interpreted from an engineering perspective to ensure their validity as solutions to realistic problems. Algorithm performance trends are then analyzed, showing that no single algorithm outperforms the others on every problem. Those that achieve the best solutions are often computationally expensive, and those that converge quickly often arrive at poor solutions. Some problem features, such as the numbers of design variables and constraints, the structural type, and the nature of the objective function, correlate with algorithm performance.

This knowledge and the generated data set are then used to develop techniques for automatic selection of optimization algorithms, based on a range supervised learning methods. Compared to a set of current, manual selection strategies, these techniques select the best

algorithm almost twice as often, lead to better-quality solutions and reduced computational cost, and—on a randomly-chosen set of mass minimization problems—reduce average material use by 9.4%.

The dissertation concludes by outlining future research on algorithm selection, on integrating these techniques in design software, and on adapting structural optimization to the realities of design.

Keywords: *Algorithm selection, structural optimization, structural design, machine learning*

Thesis Co-Supervisor: Jerome J. Connor
Title: Professor of Civil and Environmental Engineering

Thesis Co-Supervisor: John A. Ochsendorf
Title: Professor of Building Technology and Civil and Environmental Engineering

Acknowledgments

I am grateful to my advisors at MIT, Professor Jerome Connor and Professor John Ochsendorf, for their support and guidance over the past five years. Similarly, I highly value and appreciate the insight and advice of my thesis committee members, Dr. George Kocur and Dr. Abel Sanchez.

Several organizations have supported me financially and logistically in this work. I am grateful to MIT's Department of Civil and Environmental Engineering, to Mr. Robert Thurber, to the estates of Mr. Edward H. Linde and Professor Marvin E. Goody, to Professor Robert D. Logcher, and to the Singapore-MIT Alliance.

To Deirdre and Conor Clune: Thank you for all the support, encouragement, and opportunities.

Rory Clune
Cambridge, MA

Table of Contents

Chapter 1

Introduction

Structural optimization uses algorithms to seek an optimal solution to a mathematical representation of an engineering design problem. The field has been widely studied for decades, and the resulting, rich academic literature shows its potential to dramatically improve design processes and outcomes. It has seen considerable application in the aircraft and automotive design industries, with leading companies retaining expert optimization groups. By automating the cumbersome and time-consuming process of relatively late-stage design exploration, optimization allows engineers to focus on high-level design issues and to deal with complex systems whose design spaces exceed the limits of human cognition. By methodically searching a wide range of designs, optimization can lead to lighter, cheaper, and more efficient design outcomes.

Despite this demonstrated potential, extensive study, and application in certain domains, structural optimization has had little impact in civil engineering design. There are strikingly few examples of buildings, bridges, and other infrastructure designed using optimization, even though efficient and cost-effective design of such resource-intensive and expensive projects is of great importance to society.

Many of the reasons for this lack of application stem from the optimization community's failure to sufficiently address realities of practice; a lack of understanding of optimization's true capabilities and a scarceness of design engineers capable of implementing optimization techniques also contribute. A major issue is the requirement for engineers to choose, from a wide and disparate range, an algorithm to solve the design problem at hand. This choice, as the dissertation shows, significantly affects the computational expense of the optimization process and the quality of its results. Lacking the necessary expertise and experience, and in the absence of suitable guidance from the optimization community, engineers are unlikely to gain the benefits of good algorithm selection. Although they possess a well-populated optimization toolbox, engineers have no way of reliably knowing

which tool to use for the design problem at hand.

This dissertation has three distinct goals, all of which aim to narrow the gap between academic optimization and practical structural design. The first is to investigate how well a range of algorithms solve a representative set of design problems, presenting a clearer picture of optimization's capabilities than exists in the contemporary literature and demonstrating the importance of choosing algorithms appropriately. The second goal is to explore and understand the relationship between features of design problems and performance of optimization algorithms (either computational cost or solution quality), enabling the development of broad guidelines on algorithm use. Finally, we aim to develop automatic algorithm selection techniques, enhancing the applicability of structural optimization in real-world design.

This chapter presents the problem in greater detail, motivating these research goals. §1.1 discusses the potential of optimization to impact structural design—emphasizing the author's previous work on interactive optimization software—and the primary barriers to optimization's application. To test the assumption that algorithm performance varies across problems (and, hence, that the algorithm selection problem is an important one), §1.2 presents the results of a short study empirical study. After summarizing in the form of a clear problem statement and a set of precise research questions, §1.3 concludes by outlining the remainder of the dissertation.

1.1 Optimization's promise and limitations

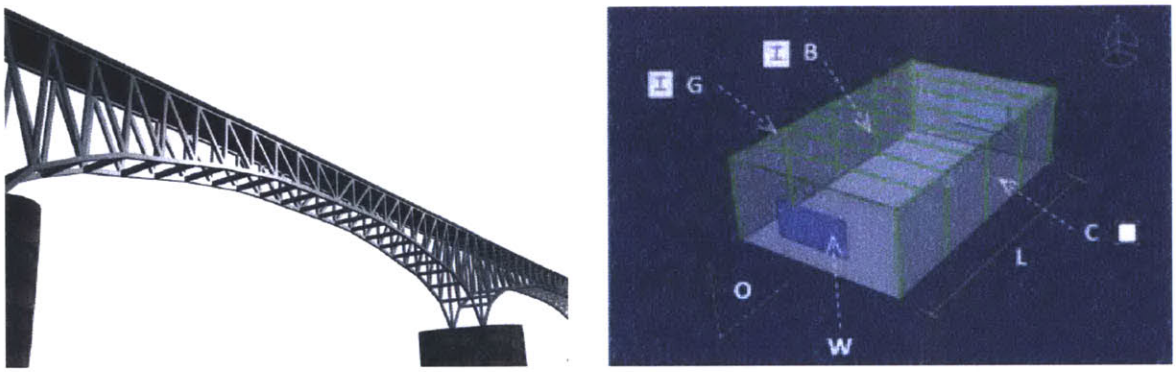
1.1.1 Potential to improve design outcomes and processes

Structural optimization's rich academic tradition spans more than a century, from Michell's seminal work on material efficiency to the more sophisticated theory of recent decades [Michell, 1904; Rozvany and Prager, 1989; Hajela and Lee, 1995]. The literature, examined in Chapter 2, consists of thousands of journal papers and hundreds of books detailing methods and applications, and many review articles have analyzed progress and identified future research directions [Venkayya, 1978; Topping, 1983; Haftka and Gürdal, 1992; Rozvany, 2009]. This wealth of information supports optimization's long-envisaged potential to significantly improve design outcomes for virtually all structural types, leading to structures that require less material, energy, and money to build and operate.

Optimization can also improve the effectiveness and efficiency of the design process, with algorithms evaluating many more alternatives than humans in a given period of time [Flager et al., 2009]. Modern computational design techniques are cumbersome and time-consuming; engineers transfer data across a set of disparate software applications, and

even minor changes to a large-scale structural model can take many hours to implement and evaluate [Clune et al., 2012]. By automating much of this process, optimization allows engineers to increase their focus on higher-level tasks such as structural system selection, integration of other technical disciplines, and improvement of constructability.

In recent years, optimization researchers have dealt with increasingly realistic representations of structures. The two-dimensional trusses and frames commonly studied throughout the field's history have, with increasingly frequency, been replaced by three-dimensional real-world examples and by multi-physics representations (Fig. 1.1).



(a) **(b)**

Figure 1.1 – The two-dimensional trusses and frames studied during structural optimization's early years have, in many cases, been replaced by **(a)** realistic three-dimensional structures and **(b)** multi-physics representations of buildings. (Images from **(a)** Smith et al. [2002] and **(b)** Flager et al. [2009].)

The development of interactive software is another recent advance. In particular, Clune et al.'s [2012] work demonstrates how the appropriate integration of optimization in the design workflow allows engineers to combine their intuition and experience with optimization's search power. The two-dimensional truss software developed in that work embeds sizing and geometry optimization capabilities in a real-time analysis environment (Fig. 1.2). Designers control problem formulations graphically, run algorithms at their discretion, modify start and end points in response to real-time evaluations, and intervene in the optimization process.

The extension of this approach to structural types beyond trusses and the integration of sufficiently robust optimization components for practical use, however, create challenges. One such challenge is the unpredictability of algorithm performance. An algorithm that works well for the two-dimensional trusses in the cited work often fails to make similar progress with, for example, bending-governed beam structures. Even within the chosen domain of trusses, slight perturbations to the problem formulation or to the starting point dramatically affect optimization algorithm performance. Addressing these, and other,

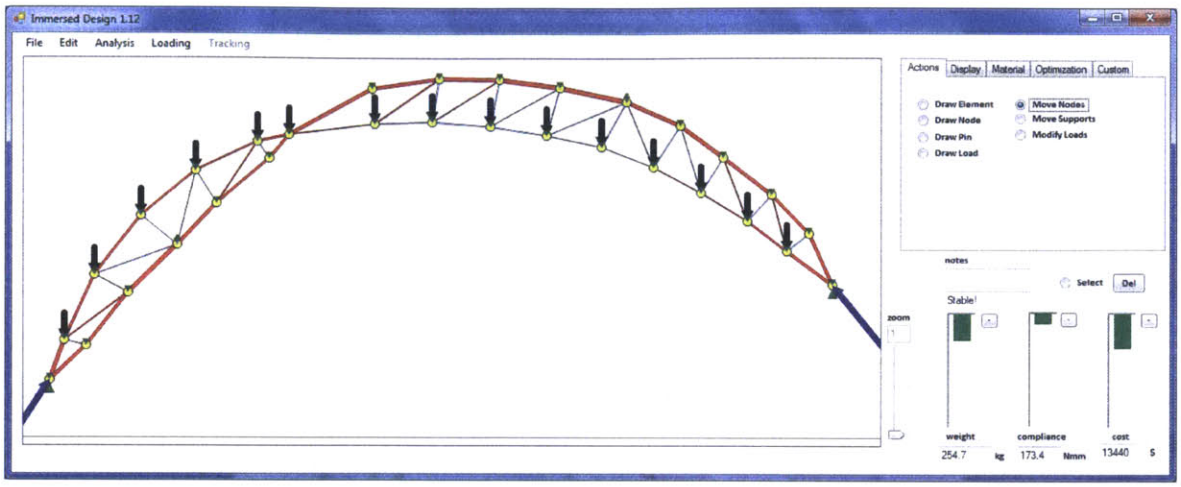


Figure 1.2 – Interactive optimization software can overcome key limitations of the field. (Image from Clune et al. [2012].)

challenges would move optimization closer to fulfilling its long-held potential. The work presented in this dissertation, therefore, is part of a broader effort to make interactive structural optimization practically useful in the design of a diverse range of realistic structural types.

Assuming the existence of an appropriately-formulated problem, a suitably-chosen and tuned algorithm, and a designer capable of setting everything up, optimization has significant potential to produce better structures while consuming less of engineers' time. As suggested in the opening paragraphs, however, these assumptions are almost certain not to hold, creating a barrier between optimization and the design industry. The following section outlines the most significant factors responsible for this.

1.1.2 Practical limitations

Despite a body of literature containing over 5000 journal papers, optimization has played a role in the design of very few structures. Researchers have proposed many reasons for this lack of application; we examine the following important ones:

a) Problem statements do not capture the full richness of a system.

b) Optimization does not sufficiently account for engineering judgment and intuition.

c) Academia and practice have fundamentally different approaches to optimization-driven design.

d) Optimization's true capabilities have not been sufficiently demonstrated.

e) Engineers are not optimization experts.

a) Problem statements do not capture the full richness of a system

Cohn [1994] points to the widening gap between professional designers and optimization researchers, emphasizing the "*irrelevance of many optimization approaches and the conflicting opinions of experts on various approaches to problem-solving.*" That paper highlights the difficulty of accurately representing the complexity of an entire engineering system as a major barrier between academia and practice. Optimization techniques then pursue the globally-optimal solution to a problem statement that does not represent many of the important realities of the underlying problem. These realities include aspects of physical behavior that lie outside the scope of the chosen modeling platform, design considerations from other engineering disciplines, and implications for construction sequencing.

The contemporary literature, reviewed in Chapter 2, has begun to address this challenge, but it remains a barrier to real-world application.

b) Optimization does not sufficiently account for engineering judgment and intuition

A number of researchers, such as Vanderplaats [1984] and Haftka and Gürdal [1992], have challenged the role of numerical optimization as a stand-alone problem-solver. The latter reference notes optimization's inability to account for the inherent uncertainty and subjectivity of real-world design, and advocates the inclusion of engineering judgment: "*... no general method of optimization, no matter how robust and powerful, can be used as a substitute for good engineering intuition. The best strategy is one that emphasizes both as complements of each other.*"

Others have addressed this limitation by developing interactive human-computer optimization tools such as the one described briefly in §1.1.1 [Von Buelow, 2008; Clune et al., 2011, 2012]. These approaches, rather than developing new optimization algorithms, seek to alter the nature of the interaction between the user and the algorithm, giving the user greater control over optimization. By seamlessly embedding optimization in the design process, with easily-modifiable start and end points; by allowing designers to control problem formulation in a graphical and intuitive manner; and by communicating information on the algorithm's progress and allowing users to interrupt and interfere, they aim to inject designers' intuition and judgment into an optimization-driven design process, accounting for the uncertainty and subjectivity inherent in structural design.

The cited research demonstrates the benefits and promise of interactive optimization, but the approach is not yet widely used by engineers.

c) Academia and practice have fundamentally different approaches to optimization-driven design

Perhaps the most important limiting factor is the fundamental difference in the approaches of academia and of professional practice. Researchers in the field have described this difference, and the author has often encountered it in practice and in conversation with engineers [Cohn, 1994].

Academic researchers seek to develop and improve mathematical algorithms; proof of success involves demonstrating strong algorithm performance on a small number of case studies, which the researcher is usually free to choose. Structural designers, by contrast, start with a design problem and look for a solution method. Before optimizing, a designer must formulate a mathematical problem statement—including design variables, constraints, and objective functions—which encapsulates the most relevant aspects of the design problem, and must then select an algorithm to search the design space for the optimum.

Academic optimization is, in this sense, a *solution-seeks-problem* approach, where the goal is to develop novel algorithms, and identifying problems for them to solve is an afterthought. This approach is evident throughout the literature, with several researchers criticizing how the community evaluates new techniques [Gent et al., 1997; Eiben and Jelasity, 2002]. Practical design, on the other hand, is driven by a *problem-seeks-solution* approach [Cohn, 1994]. Using optimization in this setting requires a set of actions which structural designers lack the expert-level insight and knowledge to perform. These actions, namely the translation of a design problem—with all its complexity and uncertainty—to a tractable mathematical optimization problem, and the selection and setup of an appropriate algorithm, demand an unrealistic level of expertise from practicing engineers.

This is a pressing and insufficiently-addressed challenge for the field of structural optimization, and a strong general motivation for this dissertation.

d) Optimization's true capabilities have not been sufficiently demonstrated

Papers in the structural optimization literature typically conclude a presentation of an optimization algorithm with a few case studies to demonstrate success. There are a few established benchmark problems [e.g., Rozvany, 1998], although researchers are usually free to choose and formulate any case study. The literature contains thousands of these successful case studies, but researchers rarely publish cases where their techniques failed to produce high-quality designs. Nor do most researchers comment on the effect of algorithm

tuning parameters and of slight code adjustments on the results, or on the extent to which the optimization algorithm is tuned to match a particular case study.

This has created a situation where the discerning engineer cannot tell, from the literature, how reliably optimization can be used as an 'off-the-shelf' generator of solutions to realistic problems without requiring expert intervention or significant trial and error. In making the case for increased usage of optimization in design, this limitation of the field needs to be addressed.

e) Engineers are not optimization experts

If its techniques are to fulfill their potential to impact the design industry, the optimization community must address the disconnect between academia and practice created by their differing approaches. Doing so requires researchers to address a number of challenges, with the goal of facilitating the use of optimization techniques by non-expert engineers. Helping engineers to suitably formulate problem statements that capture their design intent and the essential features of the structural system is one such challenge. So too are choosing an algorithm to solve the formulated problem and communicating the algorithm's progress and difficulties to the user.

Rather than seeking to integrate optimization techniques in engineering curricula or trying to convince engineering firms to hire optimization experts, we believe the optimization community's best chance to impact the design of large-scale civil infrastructure is to adapt its techniques to enable their use by non-experts. This dissertation addresses an important element of this: helping engineers to choose an algorithm for a given design problem.

To reinvigorate itself, structural optimization needs to refocus

These five factors reflect, in one way or another, the optimization community's failure to address key realities of a problem domain to which its techniques have much to contribute. As algorithmic development and refinement continue to outpace steps towards addressing reality, structural optimization remains a predominantly academic exercise. Rather than continuing to develop and refine algorithms, the optimization community should refocus a significant part of its efforts on addressing these limitations and on adapting its techniques to practice.

In describing the limitations above, particularly the differing approaches of the two fields and the need to guide algorithm selection, this chapter assumed that algorithms perform differently on different design problems. Rather than simply examining the literature and claiming algorithm selection as an important problem with a significant impact on

optimization-based design, this importance should be investigated empirically. The following section, therefore, studies the influence of algorithm selection on the solution quality and on the computational cost of structural optimization.

1.2 Variation in algorithm performance and the importance of algorithm selection

The preceding sections of this chapter, in motivating algorithm selection as an important problem, implicitly assumed variation in algorithm performance across design problems to be substantial. After all, if algorithms performed more or less similarly on all problems, the choice of algorithm would be of little importance, obviating the need to address the problem.

Although the implications of Woplert and Macready's *No Free Lunch Theorems*[1] [Wolpert and Macready, 1997] and an examination of the optimization literature (see Chapter 2) provide theoretical and anecdotal support, an original, investigation to demonstrate the importance of appropriate algorithm selection is warranted.

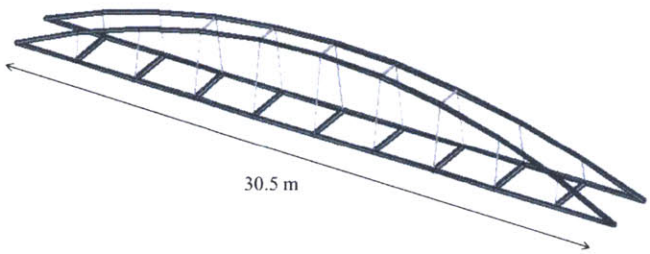
This section presents a short empirical test of the variable performance assumption, whose validity is crucial to the development of a precise problem statement and to the identification of suitable research questions. We solve three realistic design problems with eight widely-used optimization algorithms, and explore how two chosen performance measures vary across algorithms and problems. In the interest of brevity, this section briefly summarizes the problems and algorithms; §3.2.1 discusses the algorithms further, and Appendix A provides details of the problems and full results.

1.2.1 Selected design problems, algorithms, and performance measures

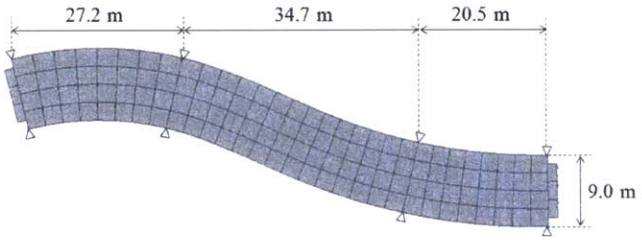
Design problems: The goal of the three design problems is to determine the minimum mass for a bridge subject to standard loading and design criteria, following the American Institute of Steel Construction's Load and Resistance Factored Design (LRFD) code [American Institute of Steel Construction, 2011]. The GSA structural analysis environment is used to analyze the structures [Oasys, 2012].

[1]The *No Free Lunch Theorems* state that, averaged over the theoretical set of all possible optimization problems, all algorithms have the same expected performance, and an algorithm's relatively strong performance on one problem is offset by relatively poor performance on another.

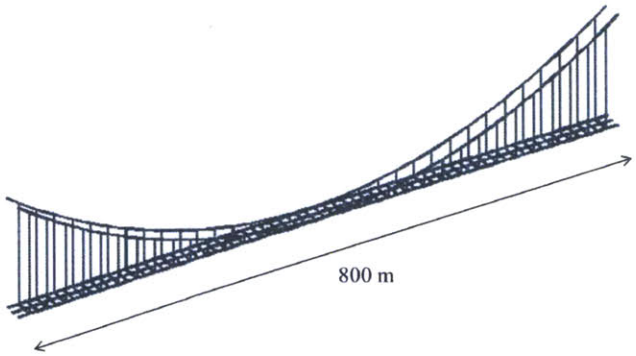
In designing two of the bridges—a 30.5 m steel arch bridge and an 800 m suspension bridge (Figs. 1.3a and 1.3c)—the algorithms vary the geometry of the superstructure and the size of the structural elements. Design solutions for the third bridge—a concrete footbridge with a 35.8 m main span (Fig. 1.3b)—vary only in the cross-sectional dimensions of the concrete beams. The problems also differ in terms of the number and type of design variables and constraints. For each problem, we formulate a continuous, non-linear, constrained optimization problem, and manually generate a feasible starting point, as



(a) The first problem is a mass minimization of 30.5 m steel basket-handle arch bridge, intended for pedestrian use.



(b) The concrete footbridge, shown here in plan view, has constant geometry, making its mass minimization a sizing optimization problem.



(c) The suspension bridge used for the third problem is much longer that the other two bridges, and its structural behavior is more sensitive to slight changes in geometry.

Figure 1.3 – The three structures on which the design problems are based: (a) a steel arch bridge, (b) a concrete footbridge, and (c) a suspension bridge

Appendix A details.

Algorithms: Eight algorithms, shown in Table 1.1, are chosen from the *NLopt* nonlinear optimization library [Johnson, 2012]. They include local algorithms, which terminate when they converge on a locally-optimal value of the objective function, and global algorithms, which broadly search the design space for a prescribed period of time, and are more likely to find the globally-optimal solution to a problem. Deterministic and stochastic methods, and methods based on sequential approximations of the design space, are used, representing the major classes of derivative-free nonlinear solvers.

Some algorithms solve only unconstrained optimization problems. In these cases we combine the objective and constraint functions in a single *Augmented Lagrangian* formulation [Conn et al., 1991]. Since these algorithms also generate the dissertation's main results, a more detailed description is deferred to Chapter 3, §3.2.1.

Table 1.1 – Eight derivative-free, nonlinear algorithms are used to solve each design problem.

Local	Constrained Optimization by Linear Approximation–*COBYLA* [Powell, 1994]
	Bounded Optimization by Quadratic Approximation–*BOBYQA* [Powell, 2009]
	Nelder-Mead Simplex—*NEL-MEAD* [Nelder and Mead, 1965]
	Subplex—*SUBPLEX* [Rowan, 1990]
	Principal Axis Method—*PR-AXIS* [Brent, 2002]
Global	Dividing Rectangles Method—*DIRECT'* [Jones et al., 1993]
	Controlled Random Search—*CRS* [Kaelo and Ali, 2006]
	Improved Stochastic Ranking Evolution Strategy—*ISRES* [Runarsson and Yao, 2005]

Performance measures: An algorithm's performance in solving the design problems can be evaluated using a wide variety of measures, which fall in one of two categories: those related to the result of the optimization process and those related to the process itself, such as the time or computational resources required to converge on an answer. This work uses the final value of the objective function and the number of calls made to the GSA analysis software as performance measures. Since the algorithms seek minima of the objective functions in each of the problems, an algorithm outperforms another if it attains a lower objective value or requires fewer analysis calls.

In practical optimization problems, most of the computational cost is associated with structural analyses. Reducing the number of calls to the analysis software, therefore, proportionally reduces the time required to get an answer—an important consideration

for practicing engineers.

Comparison of algorithms across problems requires normalization to account for the differences in the minimum attainable objective between problems and for the different number of analysis calls required to optimize over different mathematical functions. A relatively straightforward expression of the normalized performance, $\bar{p}_{a,j}$, of algorithm a on problem j is:

$$\bar{p}_{a,j} = \frac{p_{\max,j} - p_{a,j}}{p_{\max,j} - p_{\min,j}} \qquad a \in \mathcal{A}; \qquad j = 1..3; \qquad 0 \leq \bar{p}_{a,j} \leq 1 \tag{1.1}$$

where $p_{a,j}$ is the observed performance of algorithm a on problem j, and $p_{\max,j}$ and $p_{\min,j}$ are the worst and best observed performances among all algorithms on the same problem j, and $\mathcal{A}$ is the set of algorithms in Table 1.1. Normalized performance is calculated separately for each of the three design problems. The same normalization strategy is used for both chosen performance measures, so $\bar{p}$ and p could refer to either the number of analysis calls or the objective value.

1.2.2 Results and discussion

The right-hand plots in Fig. 1.4 (colored blue, or darker in grayscale) show the number of analysis calls required by each algorithm to converge on an answer for each of the three design problems. There is much variation in this measure of algorithm performance within each problem, with some algorithms outperforming others by orders of magnitude. The left-hand side of the figure shows the performance of the same set of algorithms on the three problems, now measured in terms of objective value attained. Since these are minimization problems, lower objective values always indicate better performance. Appendix A gives the values of the design variables at convergence and specifies performance measures precisely.

Fig. 1.5 directly compares algorithm performance across the three design problems, using the normalized measures of algorithm performance (Eq. 1.1). In both cases, lower values represent better normalized performance.

A clear relationship between algorithm selection and performance does not emerge from the data, but some broad trends are apparent. The three global algorithms (DIRECT, CRS, and ISRES) generally require many analysis software calls to accommodate their evaluation-intensive approaches. On the suspension bridge design problem, where the steep local gradients that characterize the mathematical space are likely to cause difficulties for some algorithms, NEL-MEAD and SUBPLEX produce the best solutions, although

they require relatively many analysis calls. COBYLA and NEL-MEAD consistently perform well in terms of objective value attained, although both are often outperformed in terms of the number of analysis calls required to achieve similar designs.

In comparing the normalized performance of each algorithm across the design problems, Fig. 1.5 demonstrates significant variation, indicating that an algorithm that performs well on one design problem may perform quite poorly on another. Some algorithms

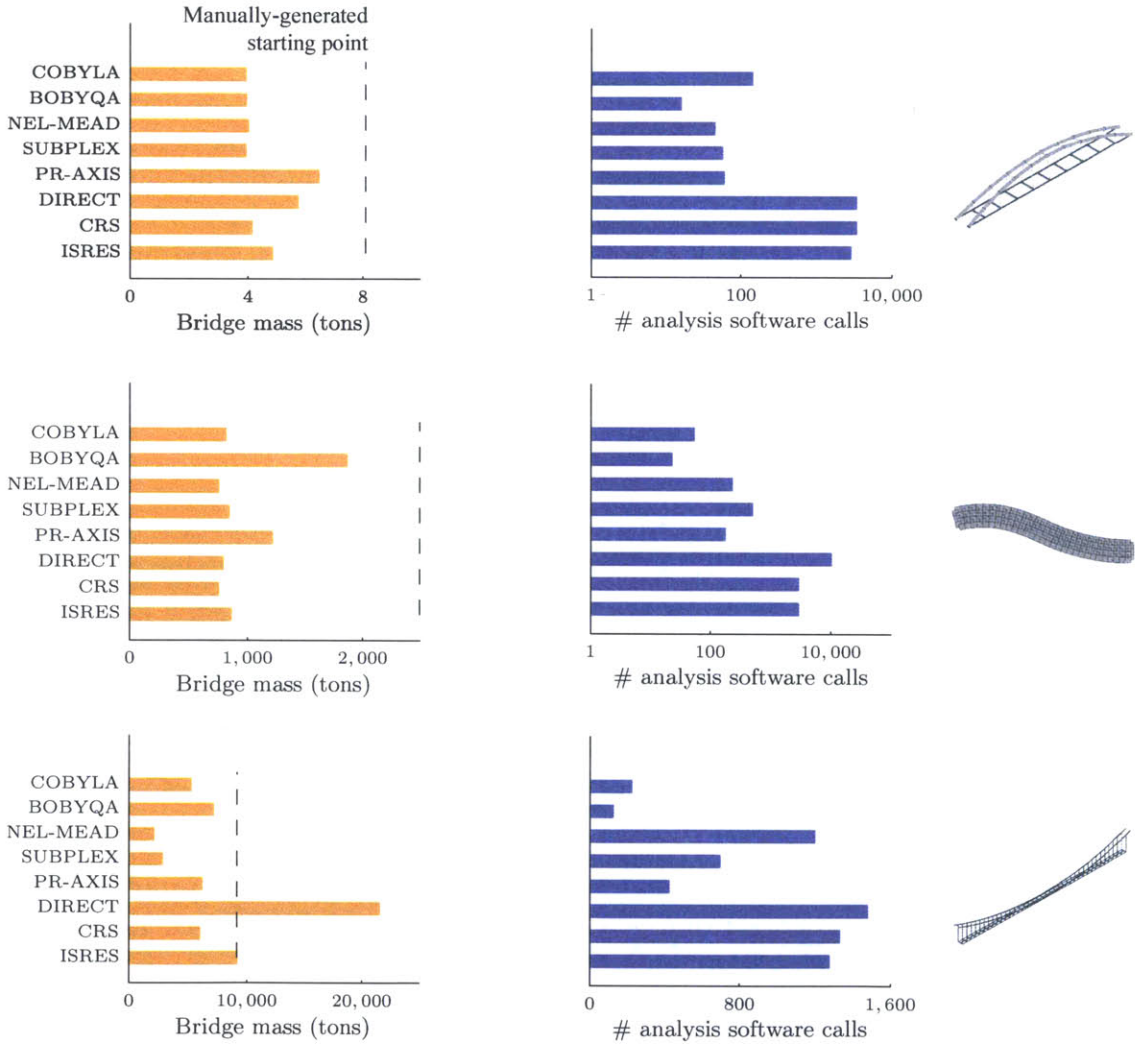


Figure 1.4 – The eight algorithms perform very differently on the three design problems, in terms of both solution quality (left hand side) and computational cost (right hand side). Algorithms that perform well on one problem do not necessarily perform well on another, and different algorithms achieve very different results for each problem.

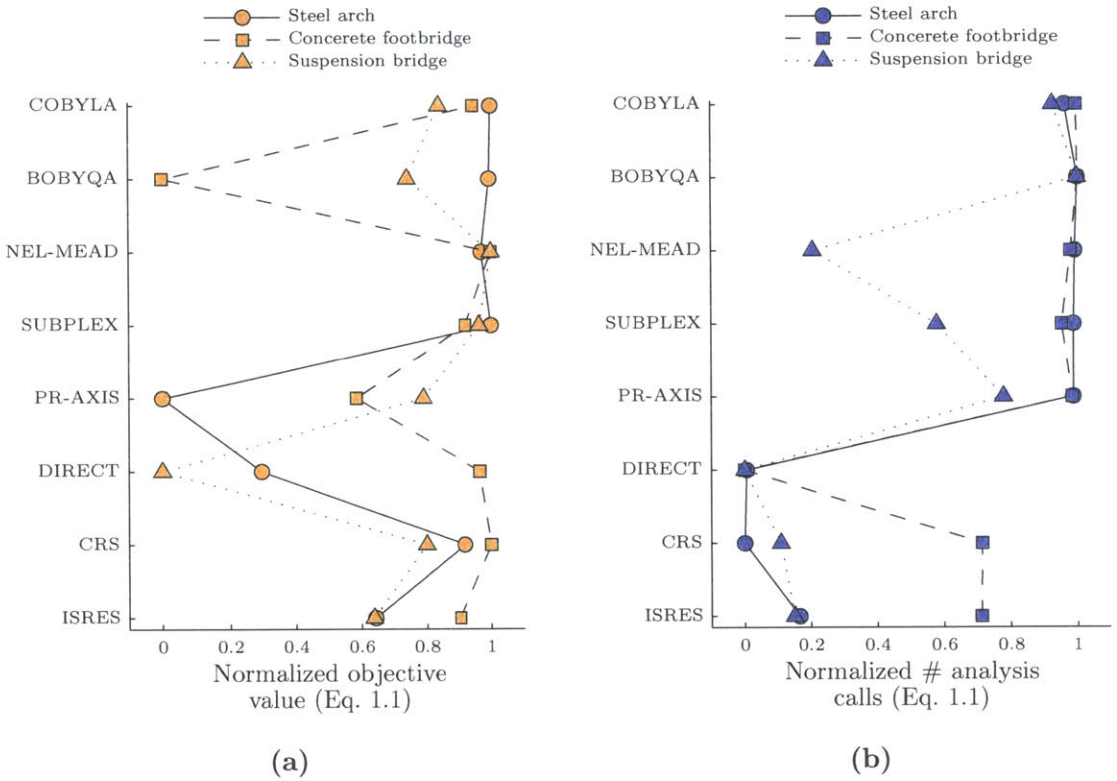


Figure 1.5 – Comparing normalized performance measures across all three design problems shows significant variation; algorithms perform well on some problems and poorly on others.

demonstrate less variation in performance than others, and many algorithms which exhibit low variation on one performance measure show high variation on the other. The results are consistent with the No Free Lunch Theorems' important conclusion that no single algorithm outperforms all others.

We could have invested more effort in tailoring problem formulations to match individual algorithm characteristics and in tuning algorithm parameters to improve performance. This, however, would not have represented the reality of using optimization in practice. Unlike optimization researchers, professional designers rarely have the expert knowledge required to perfect the mathematical problem statement or the time to repeatedly solve problems in search of the best tuning parameter values.

1.2.3 Significance

Most importantly, this study demonstrates that the choice of optimization algorithm non-linearly influences solution quality and computational cost on a set of realistic structural design problems. This provides empirical support for the previously theoretical and bibliographic case for studying the algorithm selection problem, and further motivates the stated research goals.

1.3 Problem statement and dissertation outline

Structural optimization, as evidenced by its rich academic literature and application in certain domains, has the potential to dramatically improve design process and outcomes in civil engineering. Despite this promise, optimization is virtually unused in contemporary structural design. This dissertation addresses two of the key barriers to optimization's successful application identified in the chapter: the insufficient demonstration of optimization's true off-the-shelf capabilities and the lack of guidance for engineers in selecting among the many available algorithms. This leads to the following three research questions.

Question 1: How do different optimization algorithms perform on a representative set of realistic structural design problems?

Structural optimization papers usually include a few successful applications of their proposed techniques to engineering design problems. Researchers rarely report, however, any failed attempts that preceded the successful published ones, or the extent to which algorithm tuning parameters are adjusted to deliver good results.

To address this, and to present a clear, honest picture of optimization's capabilities in design, Chapters 4 and 5 examine the performance of eight optimization algorithms on hundreds of long-span bridge design problems, laid out in Chapter 3. The results show consistent patterns of design improvement and design outcomes, but performance varies dramatically depending on the chosen algorithm and the problem at hand.

Question 2: Do correlations exist between problem features and algorithm performance?

As the empirical study in §1.2 shows, selecting the right optimization algorithm leads to better design outcomes and reduced computational requirements; this is confirmed with much more extensive data in Chapter 5. Practicing structural engineers, however, are unlikely to have the requisite expert knowledge to choose algorithms advantageously, and the optimization literature does little to help them.

Chapter 5, specifically §5.3, examines the data generated in Chapter 3 in search of statistically significant correlations between design problem features and the performance of each algorithm. These correlations are used to make a set of recommendations about when certain algorithms should be used, tangibly improving the real-world practicality of optimization.

Question 3: Can we create a system to automatically select good optimization algorithms for design problems?

The guidelines produced in response to the previous question are a useful, but potentially insufficient, solution to the problem of helping engineers select algorithms based on the design problem at hand. A guideline-based approach simplifies what is likely to be a complex problem, characterized by nonlinear and interdependent relationships between problem features and algorithm performance. Such an approach also requires careful study by engineers before it can impact design.

In response to these concerns, Chapter 6 presents a set of machine learning-based computational techniques to automatically select algorithms based on features of design problems. These 'black box' techniques complement the selection guidelines, and could be easily integrated into existing optimization software.

Dissertation outline

Chapter 2 examines the relevant literature, beginning with a broad overview of structural optimization. This supports §1.1's and §1.2's characterization of the field as predominantly focused on developing and improving algorithms, with little consensus or guidance on which algorithms to use in a given problem domain. The second half of Chapter 2 reviews the algorithm selection literature, drawing from optimization and other computational fields. Besides identifying useful precedents for this work, the chapter demonstrates a clear lack of research on algorithm selection techniques for structural engineering problems.

To understand the nature of the problem and to build selection techniques, we first produce and study an extensive set of algorithm performance data. Chapter 3 details the data generation method, which involves solving hundreds of realistic and varied bridge design problems using eight algorithms representing the major types available in practice. Chapters 4 and 5 present the dissertation's main results. Chapter 4 analyzes the thousands of designs in the generated data set, to verify that they are, in fact, realistic solutions to realistic problems. Chapter 5 explores how algorithms' solution quality and computational cost vary across problems, seeking associations between characteristic problem features and performance measures. The chapter concludes with a summary of each algorithm's observed strengths and weaknesses. Building on that knowledge and in-

sight, Chapter 6 develops and evaluates techniques for automatically selecting algorithms based on features of design problems.

Chapter 7 discusses the dissertation and its contributions to solving the algorithm selection problem, locating our work in the broader context of adapting structural optimization to the practical realities of design and identifying important future research areas.

Chapter 2

Literature review

This chapter contains two main sections. The first, §2.1, analyzes the structural optimization literature at a high level. This provides appropriate context for the rest of the dissertation and, importantly, shows that many algorithms have seen successful application in several domains. The second, §2.2, analyzes the algorithm selection literature, highlighting a range of techniques from other fields and a relative lack of research for engineering optimization problems.

2.1 Structural optimization

This section's goal is not to infer the appropriate algorithm(s) for a design problem from the past literature. Rather, by demonstrating the variation in successful algorithm application, it builds on the conclusions of §1.2's empirical study, further emphasizing the need for a system to select appropriately from the wide variety of available algorithms.

Types of optimization methods

Researchers generally credit Michell [1904] with initiating structural optimization as a formal field of study [Rozvany and Prager, 1989]. His minimum-weight *Michell trusses* are still in use today as benchmarks for research on topology optimization of framed structures [Rozvany, 1998]. Fig. 2.1 shows two such solutions from Michell's paper.

Despite notable exceptions such as Heyman's seminal work on Linear Programming-based plastic design of frames and Chan's graphical techniques for optimization using appropriate strain fields, the field remained mostly dormant for the first half of the twentieth

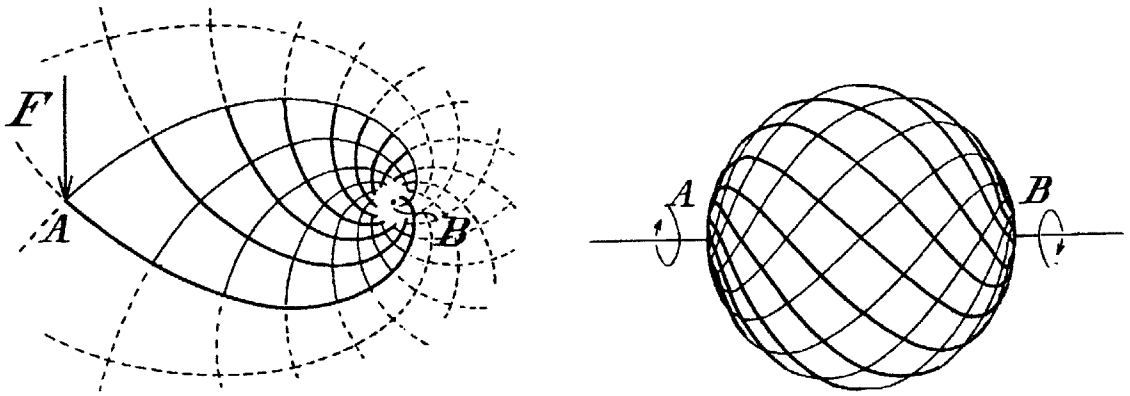


Figure 2.1 – Michell's minimum-weight solutions for **(a)** a point-loaded cantilever with a circular support and **(b)** a space frame subjected to two equal but opposite couples. (Images from Michell [1904].)

century [Heyman, 1959; Chan, 1960].

Structural optimization research then intensified and broadened in scope from the late 1960s, with the increasing availability of computing power and numerical methods. Taking inspiration from Michell's work, Prager and Rozvany [1977] developed an analogous theory for the optimum design of grillages. Hemp [1973] and Dorn et al. [1964], on the other hand, pioneered the use of pin-jointed *ground structures* to overcome limitations in Michell's theory; much of the research in the following decades has been an extension of their approach.

By the mid-1990s, researchers had published an estimated 2,500 papers and 150 books on structural optimization, and the volume of published work has since grown steadily. Several notable monographs describe the field's history and document its methods and applications [Fox, 1971; Carmichael, 1981; Siddall, 1982; Banichuk, 1983; Farkas and Pavlovic, 1984; Vanderplaats, 1984; Haftka and Gürdal, 1992; Kirsch, 1993; Adeli, 1994; Spillers and MacBain, 2009; Arora, 2011].

Structural optimization methods can be loosely classified as either *mathematical programming*, *optimality criteria*, or *stochastic search*. Mathematical programming algorithms (e.g., Powell's *Conjugate Gradient Descent* method and Nelder and Mead's simplex-based method [Powell, 1964; Nelder and Mead, 1965]) sequentially update a design based on the value of the objective function and constraints [Borkowski et al., 1990]. Optimality criteria methods start by assuming some characteristic of an optimum structure and iterate until that condition is achieved [Save et al., 1985; Rozvany and Prager, 1989]. (E.g., the *Fully Stressed Design* method assumes the optimal solution to a sizing problem is reached when all elements are at their stress limits.) Stochastic search methods, such as *Genetic Algo-*

rithms (GAs) and *Simulated Annealing* (SA), update a population of solutions based on a combination of some governing criteria and random chance [Holland, 1975; Kirkpatrick and Vecchi, 1983].

Optimization methods find optima by searching either the original design space or some approximation of it. *Approximation methods* are often useful when the design space exhibits complexity or noisy behavior in excess of what an algorithm can handle, or when function evaluations are too expensive to achieve convergence in a reasonable period of time.

Barthelemy and Haftka [1993] provide a useful review of approximation methods. Comparing two of them—linear approximation and differential equation-based approximation—Fig. 2.2 shows the errors in estimating the fundamental modal frequency of a point-loaded cantilever beam as a function of the perturbation from an initial design. The divergence in errors demonstrates the importance of carefully selecting and executing an approximation method, as the underlying design space's behavior may be quite different.

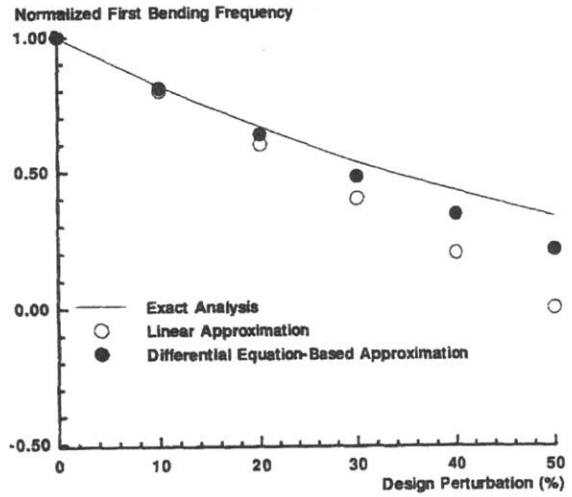


Figure 2.2 – The error in using two approximation methods to determine a modal frequency for a point-loaded cantilever beam increases with perturbation from the original design. (Image from Barthelemy and Haftka [1993].)

Addressing the realities of design

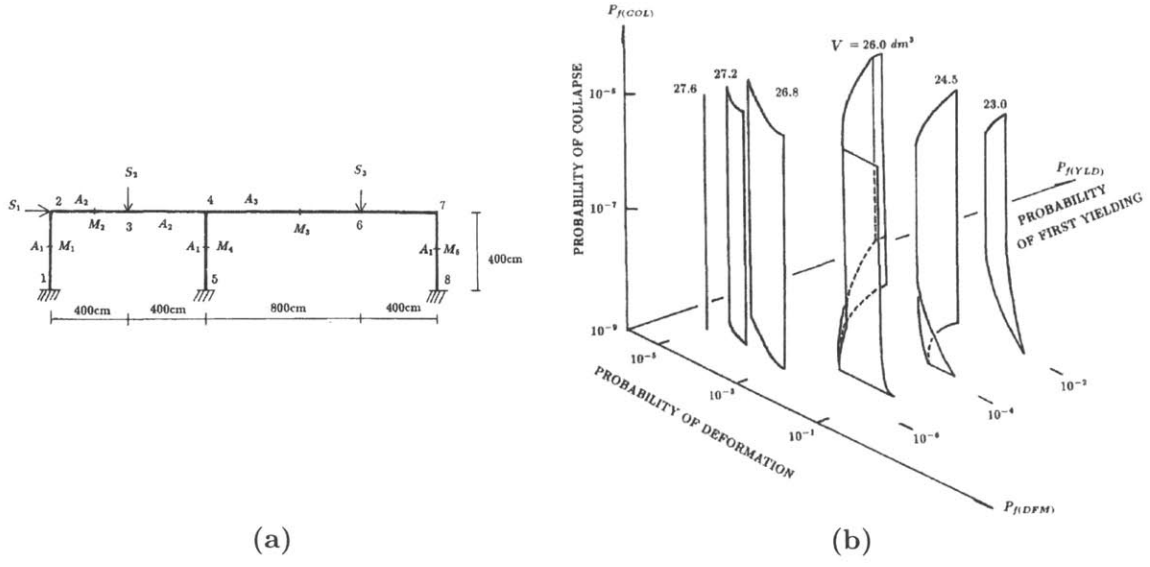
Some approaches to structural optimization, especially those from the field's early years, rely on analytic mathematical representations to predict structural behavior. To treat the realities and complexities of design, however, most recent research uses numerical analysis software—typically a commercial *Finite Element Method* solver. This leads to *black box* or *simulation-based* problem solving, where nothing is known about the objective and constraint functions apart from their values at a finite number of design points. Additional useful information, especially gradient values, is estimated using methods such as finite

difference approximation. From a practical point of view, the trade-off for this loss of information is the ability to work with realistic representations of structures.

In response to the literature's previous failure to address realistic design problems, recent years have seen an increase in the level of research addressing practical considerations. These efforts draw from diverse sub-fields, including human-computer interaction, multi-objective optimization, and multi-modal optimization.

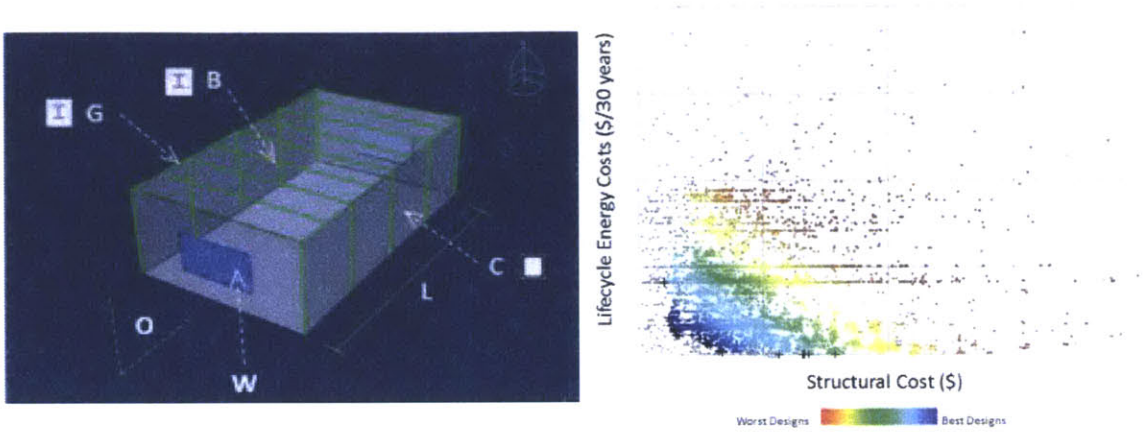
Real structural engineering problems involve more than one objective. A ubiquitous example is the requirement to balance the goal of minimum material weight or, perhaps, structural cost with the need to limit motion of the structure. To address such trade-offs, Jendo and Marks [1986], Rao [1987], and Fu and Frangopol [1988] developed some of the earliest multi-objective structural optimization techniques (e.g., Fig. 2.3).

Given structural engineers' frequent interaction with other design disciplines, diverse performance measures and design requirements should be accounted for. Flager et al. [2009] present a multi-disciplinary approach to building design (Fig. 2.4). Their approach includes structural cost and life-cycle (including energy) cost as objectives, and accounts for relevant constraints on structural performance, on the level of day-lighting, and on the available floor space.



(a) (b)

Figure 2.3 – In an early multi-objective frame example ([**a**]), Jendo and Marks [1986] plot iso-volume surfaces in a three-dimensional failure space ([**b**]), visualizing the trade-offs between four objectives: (*i*) minimizing volume, and minimizing the probabilities of (*ii*) excessive deformation, (*iii*) collapse, and (*iv*) material yielding. (Images from Jendo and Marks [1986].)



(a) (b)

Figure 2.4 – Multi-disciplinary, multi-objective optimization of a classroom ([**a**]), demonstrating the trade-off between structural and life-cycle energy costs ([**b**]). (Images from Flager et al. [2009].)

The generation of multiple candidate solutions to a problem is another reality of the structural design profession. Although these solutions can be distinguished by quantitative performance measures, engineers often select among them based on subjective or ill-defined criteria. Multi-modal optimization techniques, which seek multiple, diverse local optima within a design space, are naturally suited to addressing this aspect of design. Several researchers, including Martini [2011] (Fig. 2.10) and Balling et al. [2006], have proposed multi-modal techniques to mirror this important facet of structural design [Mahfoud, 1995].

Reacting to the inability of traditional numerical optimization to address the subjectivity and fuzziness of design—which often stems from considerations of aesthetics or constructability—Clune et al. [2011] (Fig. 1.2) and Von Buelow [2008] developed interactive optimization techniques for structural optimization[1]. (Colgan et al. [1995] gives a useful, general-purpose introduction to the concept of interactive optimization.) These allow designers to exercise control over the optimization process, leveraging their experience and intuition to offset a limitation of traditional optimization.

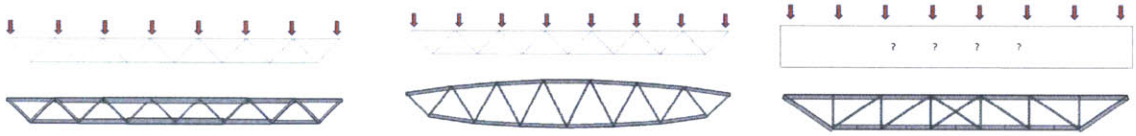
As the overall goal of this research is to enable algorithm selection for different design problems, §2.1.1 examines the types of structures regularly featured in structural optimization, identifying the most commonly-used algorithms within each structural domain. Researchers rarely publish instances of algorithm failure, so we can assume that all

[1] An alternative to accounting for subjective factors such as aesthetics by human input is to use shape grammars or other formal expressions capable of representing abstract concepts. Some of the important early work in this area was conducted by Shea and Cagan [1999].

algorithm-problem pairings presented in the literature are successful ones.

2.1.1 Successful application domains

Optimization methods address various combinations of a structure's topology (connectivity), geometry (shape), or sizing (material distribution) (Fig. 2.5). Applications in the literature range from small-scale individual components to large-scale civil structures.



(a) (b) (c)

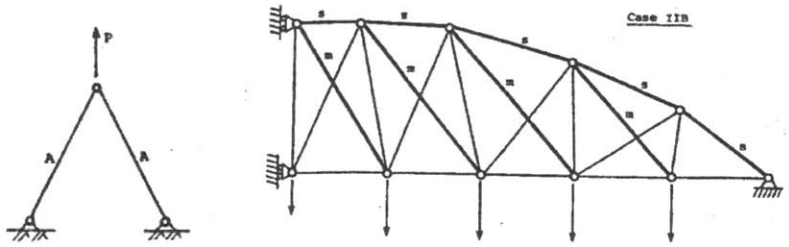
Figure 2.5 – In sizing optimization, **(a)**, the goal is to distribute material for a defined geometry and topology. Geometry optimization, **(b)**, seeks the optimum geometry of a structure whose topology is defined. Topology optimization problems, **(c)**, require a technique to define element connectivity.

Cohn and Dinovitzer [1994] review the first few decades of computational structural optimization, cataloging over 500 published optimization examples by structural type and algorithm. Their work demonstrates some interesting trends. The problems solved most extensively in the literature, for example, are far from the ones faced by practicing engineers in terms of the realism of the structure, load conditions, and limit states. In fact, deterministic optimization of simple trusses and frames accounted for 55% of all problems encountered. Examining the literature since then shows the same tendency to test algorithms on well-understood truss and frame problems.

Despite the prevalence of truss and frame examples, however, researchers have optimized many more structural types; we examine a subset of them—trusses and frames; bridges; and beams, columns, plates, and composites—below.

Trusses and frames

Topping [1983] reviews the important early work on geometry and sizing optimization (otherwise known as configuration optimization), where the geometry of a truss's nodes and the cross-sectional areas of its members are treated as continuous variables. He traces the problem's history from Schmit and Kicher's early treatment of the three-bar truss to Imai and Schmit's important Taylor expansion-based multiplier method for complex trusses (Fig. 2.6) [Schmit and Kicher, 1962; Imai and Schmit, 1981].

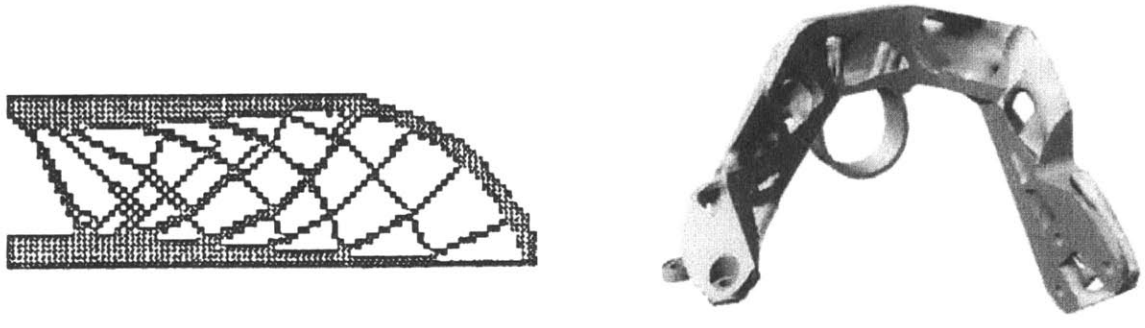


(a) (b)

Figure 2.6 – Imai and Schmit [1981], and others, extended truss geometry optimization from simple structures,**(a)**, to more complex ones, **(b)**, using a multiplier method. (Images from Imai and Schmit [1981].)

Considering only topology optimization, the various techniques fall into one of two major categories: discrete topology optimization, based on removing elements from a densely-populated ground structure; and continuous topology optimization, where material is removed from a discretization of an initially-continuous design space. Ohsaki and Swan [2002] provide a comprehensive review of the theory and applications of both.

Rozvany [2009] reviews the established continuum-based techniques, with a particular focus on *Solid Isotropic Microstructure with Penalization* (SIMP) and *Evolutionary Structural Optimization* (ESO) [Bendsøe, 1989; Xie and Steven, 1993]. Rozvany recognizes a variety of other techniques, but notes that their industrial application has not been extensive. Fig. 2.7 shows an early solution, generated by SIMP, to the often-studied



(a) (b)

Figure 2.7 – **(a)** An early SIMP-generated solution to the benchmark *MBB beam* problem, and **(b)** an ESO-generated solution to the design of an engine mount. (Images from Rozvany [2009] and Pedersen and Allinger [2006].)

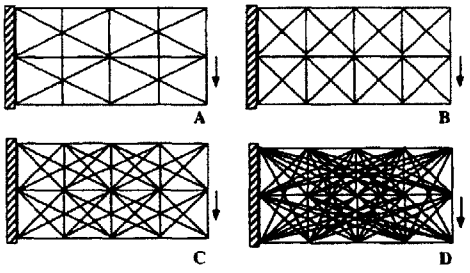


Figure 2.8 – Typical ground structures for discrete topology optimization. Elements are usually removed when an algorithm sets their cross-sectional areas below a certain threshold. (Images from Bendsøe et al. [1994]).

MBB beam problem, and a contemporary solution to an engine mount design problem, generated using ESO [Rozvany, 2009; Pedersen and Allinger, 2006].

Bendsøe et al. [1994] review discrete (ground structure-based) truss topology optimization techniques (Fig. 2.8). Their comprehensive study notes that the performance of algorithms within a given structural type depends strongly on problem formulation. An algorithm that is not well-suited to a certain weight-minimization problem, for example, may solve a minor reformulation of the same problem quite effectively.

Since trusses have long been a standard case study in optimization papers, virtually all algorithm types have been applied to them. Evolutionary search techniques, however—especially Genetic Algorithms—make up much of the recent discrete topology optimization literature. Hajela and Lee [1995] propose a two-level approach, first using singular value decomposition to eliminate the structurally unstable designs that frequently arise when using GAs for topology optimization. Rajeev and Krishnamoorthy [1997] emphasize the usefulness of GAs in handling practical issues that arise in real-world design.

Considerations of structural dynamics, usually in the form of lower-bound constraints on modal frequencies, have appeared in the truss and frame literature since they were treated in a Kuhn-Tucker approach by Lin et al. [1982] and in an unconstrained minimization formulation by Kapoor and Kumarasamy [1981]. Such constraint-based formulations continue to appear in contemporary literature [e.g., Lingyun et al., 2005; Gomes, 2011].

Bridges

Many of the truss and frame studies referenced previously, especially the longer-span ones, can be thought of as bridge design problems. In this section, we focus on the literature that explicitly sets out to address bridge optimization and the real-world challenges that arise in this domain.

One of these challenges is the selection of a structural type. Working in the domain of precast concrete bridges, Lounis and Cohn [1995] present a compromise programming-based system for choosing among optimal designs from various types (e.g., reinforced

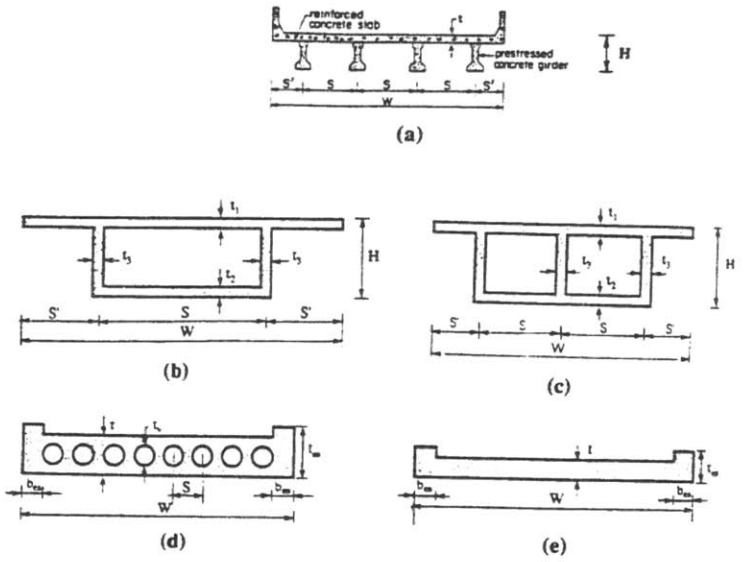
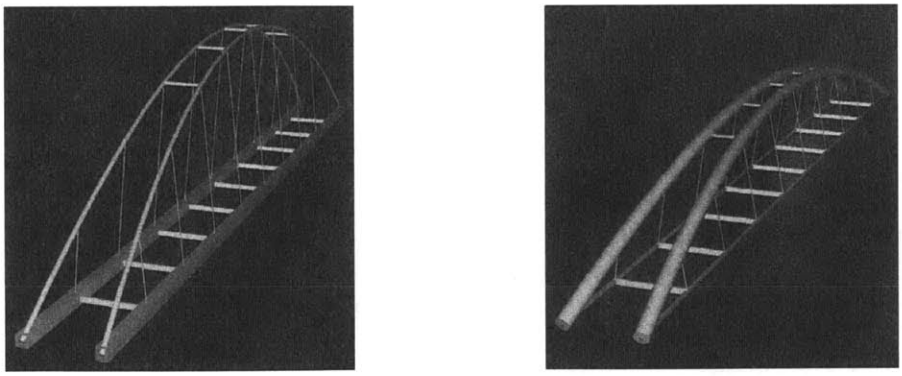


Figure 2.9 – Lounis and Cohn [1995] use compromise programming to select from different structural systems: [**a**] slab on precast I-beams; [**b**] one-cell box girder; [**c**] two-cell box girder; [**d**] voided slab; and [**e**] solid slab) for a precast concrete bridge. (Image from Lounis and Cohn [1995].)

concrete slabs, box girders, or solid slabs [Fig. 2.9]) [Duckstein, 1981].

For practical reasons, however, most research is restricted to a single structural type. For example, Guo et al. [2007a; 2007b] use various algorithms to optimize different suspension bridge topology formulations, but do not consider types such as cable-stayed bridges,



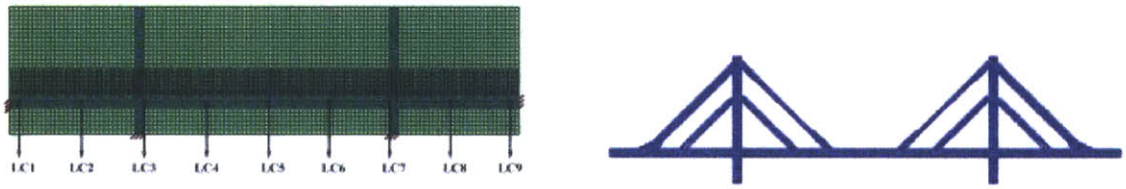
(a) (b)

Figure 2.10 – *Harmony Search* is used to find multiple solutions to a simultaneous geometry and sizing design problem for a steel arch bridge. (Images from Martini [2011].)

arches, or girder bridges. Within the domain of arches, Martini [2011] uses stochastic *Harmony Search* to generate multiple diverse designs for a basket-handle steel arch (Fig. 2.10), addressing overall geometry and element sizing.

A notable exception to the trend of searching within a single structural type is Guan et al.'s [2003] work on topology optimization of arch, cable-stayed, and suspension bridges. The authors vary the setup of the ESO algorithm and the geometric constraints on the design space to push solutions towards certain structural types. (Fig. 2.11 shows one of their examples.)

The recent trend towards incorporating realities of design is also evident in bridge optimization. Further pursuing practical considerations, Rahmatalla and Swan's SIMP-type approach to the design of long-span bridges accounts for buckling, which is of particular importance to long-span structures with compression members [Rahmatalla and Swan, 2003]. Guan et al. [2003] include lower-bound constraints on bridges' fundamental frequencies in their topology optimization work.



(a) **(b)**

Figure 2.11 – Constraining the initial design space ([**a**]) of an ESO topology optimization problem leads to the result in **(b)**. Constraints on the presence of material at certain locations in the material, and tweaks in the algorithm that encourage compression- or tension-dominated solutions, drive the divergence in structural types. (Images from Guan et al. [2003].)

Beams, columns, plates, and composites

Beam elements have long been a studied domain in structural optimization; Sheu and Prager [1968] and Chern and Prager [1970], for example, conducted some of the earliest studies on optimality criteria-based optimization of simply-supported sandwich beams. Beam and column examples appeared in 21% and 7% of the early (pre-1992) deterministic and probabilistic literature [Cohn and Dinovitzer, 1994]. Researchers have since used virtually all available methods to optimize beams, with genetic algorithms being increasingly used in recent decades [e.g., Deb, 1991; Kathiravan and Ganguli, 2007]

Plates are another structural domain whose optimization has been studied since the late

1960s, starting with the early optimality criteria approaches of Hubka [1968], Brotchie [1969], and Masur [1970]. The importance of plate design to the aeronautical and astronautical industries has largely driven this research.

Although not part of the typical civil engineer's design domain, composites are a major topic of interest in the structural optimization literature, and many of the aeronautical engineering community's developed techniques are broadly applicable beyond composite design.

Awad et al. [2012] (Fig. 2.12) review the most important and widely-studied composite optimization methods, beginning with the well-established and successful *Design Sensitivity Analysis* and *Response Surface Approximation* techniques. Researchers have also used GAs, although their population-based approach limits them to relatively small problems. Simulated Annealing performs somewhat more efficiently, and *Particle Swarm Optimization* and *Ant Colony Optimization* also show promise. The scope of the field has extended significantly since its initiation, and multi-objective and stochastic reliability problems are now reasonably common.

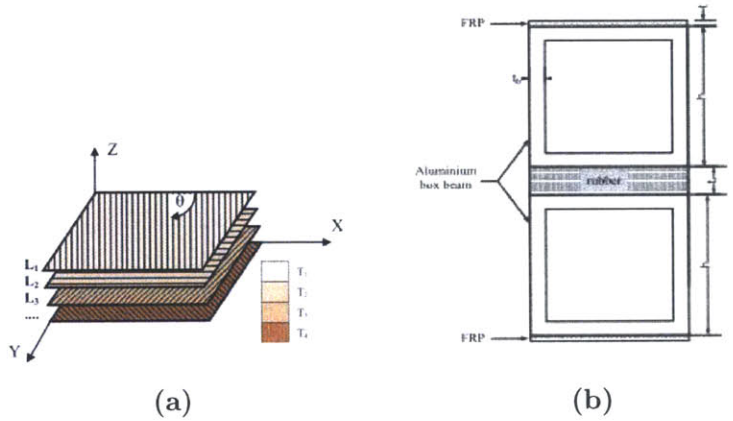


Figure 2.12 – Two types of composites commonly encountered in the literature: **(a)** a composite laminate, where the angles of orientation, θ, of each layer become design variables; and **(b)** a fiber-reinforced polymer beam with a central layer of rubber. (Images from Awad et al. [2012])

2.1.2 Summary

The existing literature contains a wealth successful applications of algorithms across many engineering domains, demonstrating optimization's potential to influence the design process. Although various algorithms have been more frequently applied than others in

certain structural domains, few identifiable patterns emerge.

As Chapter 1 noted, this potential remains unrealized due to various limitations. An important one of these is the inability of non-experts to suitably choose from among the available and successfully-demonstrated algorithms. The way in which structural optimization developed—with researchers developing algorithms and then seeking case studies on which to demonstrate those algorithms' ability—has created a situation where a designer has no methodical or reliable means to select an algorithm to solve a given problem.

Another important limitation of the literature, evident in virtually all of the referenced work, is a general tendency to report successful algorithm performance on a small number of case studies. Researchers do not typically mention the extent to which algorithms were tuned to match a particular case study, or whether the algorithm would perform as well on a large set of similar problems. This leaves the discerning engineer with an unclear picture of the field's true potential to impact design, and motivates our investigation of the performance of algorithms—with the same, default tuning parameters and setup—on a wide range of related problems later in the dissertation.

To enhance its real-world applicability, the field of optimization needs to adapt itself to the realities of the design. This dissertation takes one of many important steps towards this by addressing the algorithm selection problem.

2.2 Algorithm selection

Most algorithm selection research has focused on choosing algorithms for problems such as linear systems, partial differential equations, and data mining. The approaches taken in this literature, however, can inform the development of an optimization algorithm selection system, justifying its review here. In addition, some canonical papers, outlining general-purpose approaches to the algorithm selection problem, are identified.

Few researchers have worked on algorithm selection for optimization, however. To the best of our knowledge, no work to date has focused on selecting algorithms for structural design problems in civil engineering.

§2.2.1 presents general approaches to algorithm selection. An examination of the non-optimization literature follows in §2.2.2, and §2.2.3 concludes the chapter by looking specifically at the algorithm selection literature for optimization, identifying a gap in the development of such systems for structural engineering problems.

2.2.1 The algorithm selection problem

Rice [1976] provided the earliest formal definition of the general algorithm selection problem, presenting it as an exercise in fitting a function to map algorithms to problem instances based on a chosen set of problem features.

Fig. 2.13 highlights each of the key steps in the algorithm selection process. The first is the extraction of a set of characteristic features from the problem space. This feature set must be capable of representing the problems' mathematical structures and varying complexities while allowing algorithms' advantages and disadvantages to be exposed. The mapping $S(f(x))$ from this feature space to the algorithm space is the core of the algorithm selection system; its goal is to select the algorithm whose performance p is, in some sense, best. Smith-Miles [2008] described how the development of a selection mapping $S(f(x))$ became recognized as a meta-learning task by researchers in diverse fields of application.

Rice's high-level structure (Fig. 2.13) is a useful one to apply when thinking about algorithm selection; most, if not all, of the literature follows this general approach.

A subsequent paper by Rice [1979] expanded on the general problem definition by providing a concrete methodology for choosing an algorithm to solve numerical integration and partial differential equation problems.

This lead to the more sophisticated *PYTHIA-II* system [Houstis et al., 2000], whose system architecture is shown in Fig. 2.14. The system aims to handle the highly complex *"algorithmic discovery of knowledge from performance data and the management of these data."* Many of the features of PYTHIA-II, such as *knowledge discovery in databases* and the approach to constructing *recommender systems*, are useful precedents for the design

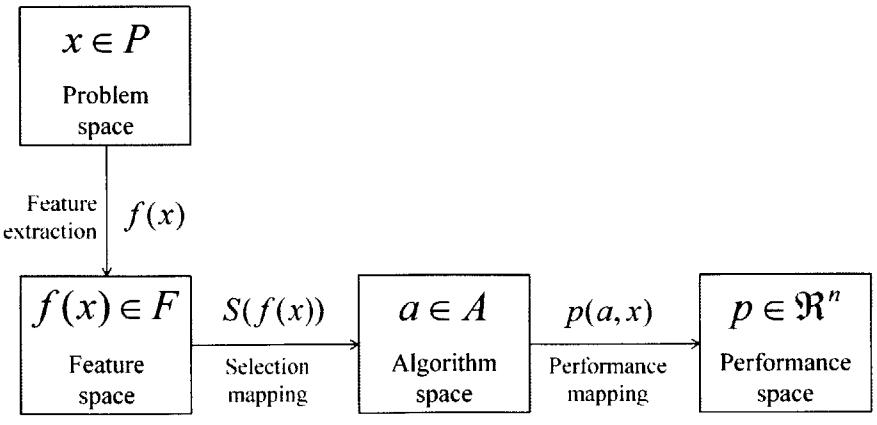


Figure 2.13 – Rice's original representation of the algorithm selection problem has formed the basis of most research in the field. (Adapted from Rice [1976].)

of a similar system for structural optimization.

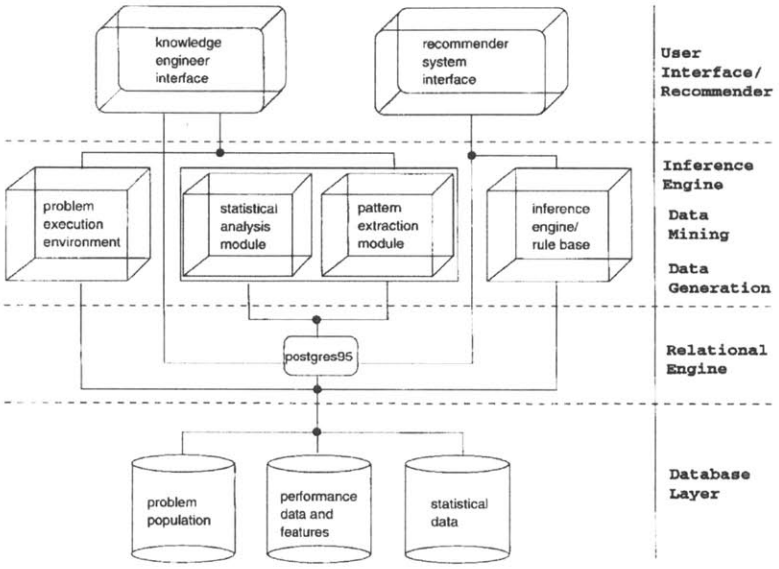


Figure 2.14 – The System architecture of the *PYTHIA-II* software recommendation system, which constructs algorithm selection rules by mining software performance data, reflects much of Rice's [1976] original framing of the problem. (Image from Houstis et al. [2000].)

2.2.2 Non-optimization algorithm selection

Substantial variation in algorithm performance across problems is by no means limited to optimization. Such variation, and the associated need for algorithm selection strategies, arises in many disciplines throughout computer science and mathematics [Smith-Miles, 2008]; the following highlights notable research in some of those disciplines.

Linear solvers and differential equations

The wide availability of multiple solvers, the relatively short times required to solve most problems, and the ease of determining suitable classification features for well-understood problem spaces make linear systems and partial differential equations (PDEs) natural candidates for algorithm selection schemes.

Many researchers have worked in this area. The system developed by Weerawarana et al. [1996], for example, incorporates user preferences regarding accuracy and solution time

to select the algorithm it expects to perform best. It does so by matching new PDEs to a population of previously-solved ones, based on a chosen set of problem characteristics.

Eijkhout and Fuentes [2010] formulate the problem as one of classification. Their approach defines a method as the combination of a pre-conditioner and an iterative solver, greatly increasing the solution space cardinality. Bhowmick et al. [2009] present a set of techniques to speed up the algorithm selection process for linear solvers by discarding low-information and redundant characteristic features.

Other researchers have contributed to this sub-field of study, and various elements of their approaches inform the development of an algorithm selection scheme for different classes of problems [Arnold et al., 2000; Dongarra et al., 2006; Fuentes, 2007].

Simulation systems

Many techniques are often available to solve a given simulation problem; Ewald et al. [2008] seek to take advantage of their varying performance levels by developing a system to choose from the library of simulation techniques in their existing *JAMES II* system. The authors emphasize the need for online learning, where the system improves its mappings with each additional observed instance, and they successfully demonstrate a decision tree-based algorithm selection scheme. The overall computational framework closely matches Rice's problem structure [1976].

Although not limited to simulation systems, Wilson et al. [2000] describe the use of case-based reasoning techniques for selecting components in general problem-solving environments. This contrasts with most of the literature, which uses inference methods to generalize beyond a given set of examples.

Data-mining algorithms

Focusing on data-mining applications, Hilario et al. [2009] extend Rice's framework to account for characteristic features of the algorithms themselves. By relying on an ontology of data-mining algorithms, this approach allows the authors' algorithm selection methods to generalize in the algorithm space, instead of the problem space-only generalization usually encountered in the literature.

2.2.3 Algorithm selection for optimization

The proliferation of optimization methods available to solve different problems, and their varying performance (supported by the *No Free Lunch* theorems and observed empirically in §1.2 and elsewhere) have lead some researchers to address the problem of optimization algorithm selection.

The first thorough approach to the problem was an effort by Mittelmann and Spellucci [1998] to build the *Decision Tree for Optimization Software*, an online repository which catalogs optimization algorithms by the types of problem to which they are suited, providing links to related technical literature and resources. This approach assumes users have sufficient domain and mathematical expertise to characterize their optimization problems.

The next step beyond cataloging is automatic extraction of useful features from problem instances, to serve as the basis for a selection system. The *Dr. Ampl* system does this for optimization problems expressed in the *AMPL* programming language [Fourer and Orban, 2012; Fourer et al., 2002]. Dr. Ampl can serve as a stand-alone feature extractor or can interface with the library of algorithms stored and executed on the *NEOS* server [Czyzyk et al., 1998].

Hough and Williams [2006] outline the use of boosted binary decision trees and support vector machines to select algorithms for generic optimization problems. Many of the problem features they use, however, relate to the mathematical structure of optimization problem instances. Computing these features for the simulation-based problems in structural engineering would require an unreasonable amount of time and computational resources. Leveraging domain-specific knowledge, on the other hand, should allow researchers to determine a set of easily-computable features that could serve in place of mathematical ones.

Fukunaga et al. [1997], as part of their broader effort to develop the *OASIS* spacecraft design environment, present steps towards an optimization algorithm selection system. After exploring a typical design space, the authors discuss the integration of domain-dependent and domain-independent knowledge to infer (using Bayesian networks) the appropriate algorithm and its tuning parameters to solve the problem at hand. They assume that stochastic algorithms work best for their class of problems, and restrict their range of algorithms to GAs, SA, or hybrids of the two.

Ong and Keane [2002] present another domain-specific approach, evaluating a set of classification techniques that select optimization algorithms for simulation-based aircraft wing design. The similarity of all the problems solved in that work, however, limit its direct applicability to structural engineering, where the relative variation in problems is greater.

Many researchers in the field emphasize the frequency with which their techniques select

the best optimization algorithms, at the expense of evaluating the quality of the resulting physical designs. Any efforts to develop selection techniques for civil engineering problems should explicitly focus on maximizing the overall quality of solutions produced by the selected algorithms.

2.3 Summary

The existing literature supports structural optimization's potential to improve design processes and outcomes, although a tendency to report algorithmic performance on a small number of specific case studies somewhat undermines the case for using optimization in general-purpose engineering design, where many different problem types are encountered.

Insufficient adaptation to the realities of design, described in §1.1.2, continues to inhibit the field's practical application. A *solution-seeks-problem* approach—where researchers focus primarily on developing and improving algorithms—is evident in the literature. Many algorithms have been successfully used in many domains, but there is no easily-discernible pattern to this and no reliable means for the optimization non-expert to choose an algorithm for a given problem. As part of a move towards addressing the realities of using optimization in design, there is a pressing need to move beyond development and demonstration of algorithms to engage with the broader issue of enabling designers to choose algorithms suitably. This is a primary goal of this dissertation.

The algorithm selection literature, in other numerical fields and—more recently—in some areas of optimization, provides useful precedents for addressing the problem. Algorithm selection has been most successfully approached with supervised machine learning.

To date, however, no researcher has developed an algorithm selection system for structural engineering problems. The wide range of problems encountered in practical structural design (explored in the first half of this chapter) limit the direct applicability of the techniques developed for other fields, and motivate a solution of the algorithm selection problem from structural engineering's unique perspective.

Chapter 3

Method for generating optimization data

The simulation-based nature of modern structural optimization and the many factors that characterize structural design problems make the algorithm selection problem a complex one. Faced with such complexity, we choose a statistical approach, deriving insights by examining how algorithms perform when solving structural optimization problems.

The first step in this approach is the identification of a suitable dataset. The subject of this chapter is the generation of such a dataset by solving a range of optimization problems with a diverse set of algorithms. The data should cover an interesting range of structures and should be large enough to enable statistically significant conclusions. On the other hand, practical considerations—especially the considerable computational cost of simulation-based optimization—limit the amount of data that can reasonably be generated. Recognizing this tradeoff, the study is limited to bridge design problems only.

§3.1, following a discussion of relevant design criteria and codes, presents 474 bridge optimization problems based on six bridge types. These are solved by the same eight optimization algorithms used in Chapter 1's empirical study (revisited in §3.2); this generates 3792 algorithm-problem pairings, each with a set of algorithm performance measures and a resulting structural design.

The second half of the chapter—§3.3 and §3.4 describes the computational infrastructure that writes and solves the problems and that stores and manipulates the resulting data.

3.1 Structural optimization problems

The 474 optimization problems used to generate the dataset are presented here. They represent realistic design scenarios which engineers encounter in practice, and are formulated using relevant aspects of modern structural design codes. We begin by presenting design considerations and computational techniques that apply to all problems. Two subsequent sections, §3.1.2 and §3.1.3, describe the problems and the structural models on which those problems are based. Further details of both appear in Appendix B.

3.1.1 Common elements of structural design

All the design problems rely on certain physical properties, rules from the American Institute of Steel Construction's (AISC) steel design code, and custom computational approaches to traditionally manual exercises. To minimize repetition and to keep §3.1.3's presentation of the design problems suitably concise, the following paragraphs describe these shared elements.

Design codes

The design problems use the *Load and Resistance Factor Design* (LRFD) philosophy (also known as *Limit State Design*), which compares the load that a structural component must carry—its *required strength*—to its *actual strength*. This work uses the 14^{th} edition of the AISC's LRFD-based *Steel Construction Manual* (SCM) [American Institute of Steel Construction, 2011].

To reflect a range of possible uncertainties, LRFD applies scaling factors to increase required strength. All load cases are multiplied by suitable ϕ factors (Table 3.1), and each element's actual strength, measured as the yield stress of its material (Table 3.2) is evaluated under a combination of live and dead loads.

The SCM mandates the consideration of various live loading patterns; a bridge's maximum vertical displacement due to any of these unfactored (ϕ factors omitted) load patterns should not exceed $\frac{1}{800}$ of the bridge's span.

Physical properties

All the bridges carry the same applied loading (Table 3.1). We do not explicitly model the bridges' road surfaces; their presence is instead represented by a uniformly-applied dead

Table 3.1 – Each bridge is required to support the same dead and live loads.

	Source	Magnitude (kPa)	LRFD ϕ factor
Dead Load	Weight of bridge superstructure	*-varies-*	1.2
Dead Load	Road deck	3.80	1.2
Live Load	Four lanes of traffic	4.10	1.6

load of 3.80 kPa. This eliminates the need for two-dimensional finite element modeling, significantly reducing model run times and allowing consideration of more optimization problems. The downside is that models underestimate the bridge decks' stiffness and structural mass.

In addition to the dead load due to self-weight and road finishes, each bridge is subjected to a uniformly-distributed live load of 4.10 kPa across its 20 m-wide deck, representing four lanes of traffic. The design problems consider two live load patterns when evaluating the displacement caused by traffic: uniform load along the entire bridge span and uniform load along one half of the span. The latter case is generally the more onerous, especially for long-span structures.

Steel and concrete are the only materials used in the study. Table 3.2 gives their relevant material properties.

Table 3.2 – All structures have the same material properties.

	Yield Strength (MPa)	Young's Modulus (GPa)	Density (kg/m^3)	Poisson's Ratio
Steel	300	200	7,850	0.3
Concrete	40	20	2,400	0.2

Continuous approximation to discrete element sizing

The design variables for all optimization problems control either the geometry of a bridge or the size of its individual steel elements. In manual structural design, sizing of steel elements is an inherently discrete process. Engineers choose a cross-section for each element from the AISC's catalog of shapes, and then check its performance against code requirements [American Institute of Steel Construction, 2011]. The set of feasible cross-sections for every structural element is therefore finite and discrete.

Most of the widely-used nonlinear optimization algorithms, however, use only continuous variables and cannot handle discrete selection problems. To adapt these algorithms to real-world design problems, we formulate a continuous approximation to the discrete sizing

problem. A continuous variable set by the algorithm maps to a cross-section which has similar structural properties to a cross-section in the AISC catalog.

To make this problem manageable, it is assumed that all structural elements in this work—apart from cables—are wide-flanged I-beams, known as *W-sections* (Fig. 3.1). Other commonly-used steel cross-sections, such as rectangular and circular hollow sections, T-beams, and channels, are omitted. W-sections are categorized by the approximate heights of their cross-sections (H in Fig. 3.1); all W16s are approximately 16 inches high, W30s measure 30 inches in height, etc. As the cross-sectional area increases within a category of W-sections, most of the additional material appears in the section's flanges rather than its web.

Adding material to the flanges (increasing T) does not change either H or B, so the relationships between A and the section's other relevant properties—I_{maj}, I_{maj}, J —are close to being linear within a single category of W-sections (Eq. 3.1).

$$\begin{aligned} I_{maj}(A) = I_{maj}(T) &= T(B\frac{(T+H)^2}{2} + \frac{1}{6}BT^2) + \frac{1}{12}tH^3 \\ I_{min}(A) = I_{min}(T) &= \frac{1}{6}TB^3 + \frac{1}{12}Ht^3 \\ J(A) = J(T) &= 2BT^3 + Ht^3 \end{aligned} \tag{3.1}$$

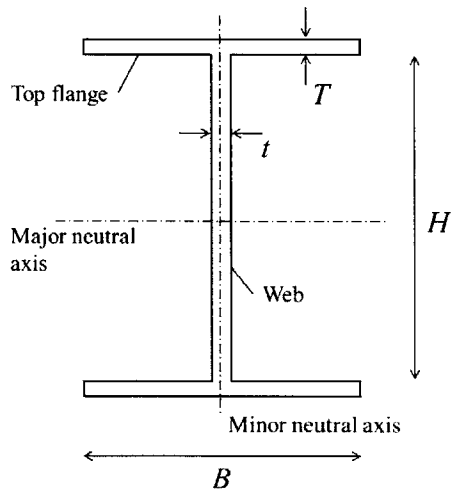


Figure 3.1 – Within a category of W-section shapes, most additional material is added to the flanges, increasing T.

where I_{maj} and I_{min} are a W-section's area moment of inertia about its major and minor neutral axes, J is its torsional moment of inertia, and the other variables are as shown in Fig. 3.1.

This linearity is exploited to fit linear models between cross-sectional area and each of the three moments of inertia within all W-section categories. Cross-sectional area is then chosen as the independent, continuous design variable to be determined by the optimization algorithms, and the linear model for the W-section category that maximizes I_{maj} is used to determine the now-dependent variables I_{maj}, I_{min}, and J.

The details of this model fitting and an evaluation of the linearity assumption's validity are given in Appendix B.1.

Measuring overall structural stiffness

The optimization problems all have either mass or structural stiffness as their objective function. Evaluating a bridge's mass is conceptually straightforward, but using stiffness as an objective function requires more careful consideration.

A stiffness objective function should return a single scalar evaluation of every point in the design space. To do so, it post-processes the structural responses from the analysis software to arrive at a single value representing overall stiffness. Many measures could be used, such as the first modal frequency or the average nodal displacement under a particular load case. This study, however, chooses the difference between the absolute maximum displacement and its upper bound—under unfactored live load applied along the entire structural span—as the objective function (Eq. 3.2).

$$f(\vec{x}) = |\Delta(\vec{x})|_{\max} - \Delta_{\text{limit}} \tag{3.2}$$

where Δ_{limit} is the allowable upper bound on displacements, $|\Delta(\vec{x})|_{\max}$ is the magnitude of the maximum structural displacement, and $f(\vec{x})$ is the amount by which the maximum structural deflection exceeds its upper bound for a design point $\vec{x}$[1]. As will be seen in §3.1.3, all the stiffness optimization problems set an upper bound on the allowable structural mass. The goal of these problems is, therefore, to find the stiffest possible structure using a fixed amount of material.

We recognize that other stiffness evaluations will produce objective functions with different mathematical characteristics to ours, and that this may limit the dissertation's conclusions about algorithm selection for structural stiffness optimization to problems

[1] Δ_{limit} being a constant, this is of course equivalent to simply minimizing $|\Delta(\vec{x})|_{\max}$.

that have Eq. 3.2 as their objective function. Nevertheless, the lack of previous research in this area necessitates such limitations; exploring the effect of variation in stiffness objective formulation is a future research task.

3.1.2 Bridge models

Six steel bridge types—basket-handle arches, trussed arches, cable-stayed bridges, girder bridges, suspension bridges, and Warren trusses—and a few suitable span lengths combine to give fourteen distinct bridge structures (Table 3.3).

The following sections presents each of these types in turn, identifying the variables that control sizing and geometry design and specifying the relevant mathematical relationships used to determine geometry. The sections examine a single model from each type; the additional ones referenced in Table 3.3 are in Appendix B.2.

In each case, we use engineering experience and rationale to determine a feasible starting point. The section specifies these starting points, along with upper and lower bounds on each design variable.

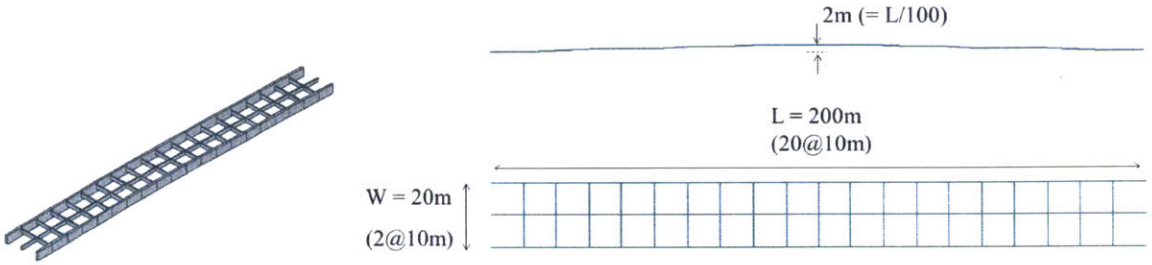
Table 3.3 – The chosen bridge types and spans yield fourteen different structures.

Bridge type	Chosen spans	Figure	Table
Basket-handle arch	50 m, 200 m, 300 m	3.4	3.4
Trussed arch	200 m, 300 m	3.5	3.5
Cable-stayed	300 m, 500 m, 800 m	3.6	3.6
Girder	50 m	3.7	3.7
Suspension	500 m, 800 m	3.8	3.8
Warren truss	50 m, 200 m, 300 m	3.9	3.9

Road decks

Before presenting the full bridge models, the following paragraphs describe their road decks, which are sufficiently similar to warrant collective discussion.

Each road deck follows one of two design concepts. The first is a single layer of regularly-spaced beams, used for structures with short spans or rigid superstructures; the second is a more rigid two-layer trussed deck, used for the long-span cable-stayed and suspension



(a) Isometric projection **(b)** Side elevation and plan

Figure 3.2 – The design of the 200 m single-layer road deck is controlled by three sizing variables: the cross-sectional areas of the outer longitudinal beams (A_1), of the inner longitudinal beam (A_2), and of the evenly-spaced transverse beams (A_3). These design variables are used by all models that have this deck configuration.

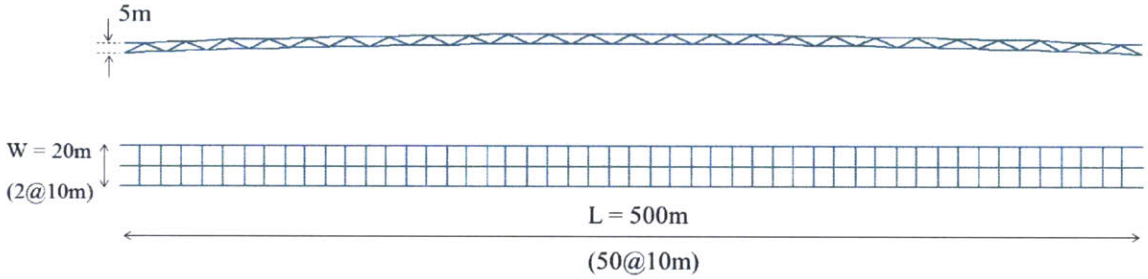
bridges, and for two of the basket-handle arches (Figs. 3.2 and 3.3). In every model, the decks have the same width (20 m) and transverse beam spacing (10 m).

The single-layer deck is composed of three continuous, longitudinal beams running between the deck's support locations. These beams are simply-supported: one end is pinned, preventing displacement but allowing rotation; the other is roller-supported, allowing rotation and horizontal displacement. The deck's element sizing is controlled by three continuous variables—A_1 and A_2 are the cross-sectional areas of the outer and inner longitudinal beams, and A_3 is the transverse beams' cross-sectional area. The additional sectional properties required for analysis are determined using the element sizing procedure in the previous section. This deck is used by all the trussed arch, girder, and Warren truss bridges, as well as the 50 m basket-handle arch bridge.

The second, trussed deck design is used for the longer-span cable-stayed, suspension, and arch bridges. It consists of two single-layer decks separated by a vertical distance of 5 m and connected with diagonal elements (Fig. 3.3). The longitudinal beams are simply-supported in the same manner as the single-layer deck's beams. The sizing of this deck is controlled by the same three variables as the single-layer deck, plus an additional variable, A_4—the cross-sectional area of the diagonal connecting elements.

Each of the bridge decks, which support distributed traffic loading and additional dead load representing road finishes (Table 3.1), has a precambered span-to-rise ratio of 100 : 1, rising parabolically from the end supports to midspan. The height of a deck spanning 300 m, for example, rises to a midspan maximum of 3 m above its supports.

(a) Isometric projection



(b) Side elevation and plan

Figure 3.3 – A 500 m simply-supported trussed (i.e., double-layer) road deck model. In addition to the same three sizing variables as the single-layer deck (Fig. 3.2), the trussed decks have a fourth sizing variable, A_4, which controls the size of the diagonal web elements.

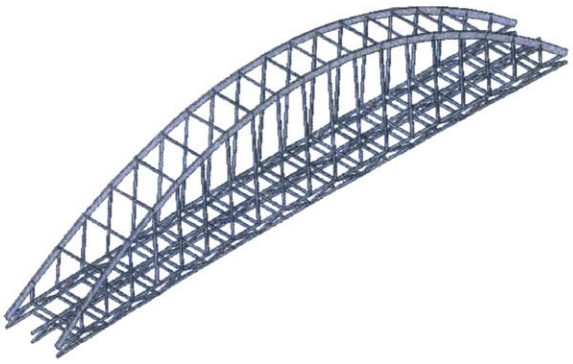
Basket-handle arch bridges

The first bridge type is the steel **basket-handle arch** (Fig. 3.4); the geometry of the arches is controlled by the independent design variables γ_z—the arches' span-to-height ratio—and γ_y—their inward lean-to-span ratio (Eqs. 3.3). In addition to the deck sizing variables, the other design variables are the cross-sectional areas of the main arch ribs (A_4), of the hangers connecting the ribs to the deck (A_5), and of the struts connecting the two arches (A_6).

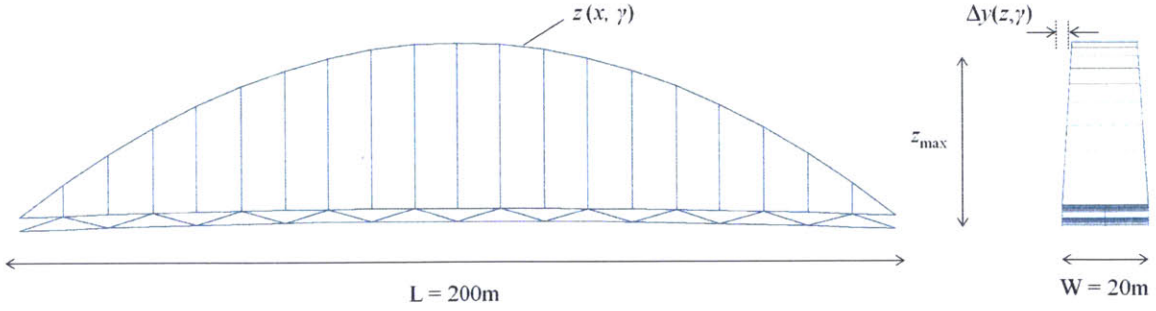
$$z(x, \gamma_z) = \frac{(x - \frac{L}{2})^2}{L(\frac{1}{\gamma} - \frac{\gamma}{4})}$$
$$\Delta y(z, \gamma_y) = \frac{\gamma_z}{\gamma_y} z \tag{3.3}$$

where x and z are the x- and z-coordinates of points along the arch (both measured from the left-hand intersection of the arch and the deck), L is the span of the arch (either 50 m, 200 m, or 300 m), W is the width of the deck (20 m in all cases), and Δy is the magnitude of the inward lean of the arch at a given height z.

Table 3.4 gives the starting values of the design variables for the three basket-handle arch models (spanning 50 m, 200 m, and 300 m).



(a) Isometric projection



(b) Side elevations

Figure 3.4 – 200 m steel basket-handle arch. In addition to the design variables controlling deck element sizing, the design vector includes the cross-sectional areas of the arch ribs (A_4), the deck hangers (A_5), and the struts connecting the ribs (A_6), as well as the span-to-height and span-to-inward-lean ratio of the arches (γ_z and γ_y).

Table 3.4 - Starting values and bounds for the basket-handle arch bridge design variables

		50 m			200 m			300 m		
		LB	Value	UB	LB	Value	UB	LB	Value	UB
$A_1(m^2)$	Outer long. deck beams	0.001	0.05	10.00	0.001	0.11	10.00	0.001	0.30	10.00
$A_2(m^2)$	Inner long. deck beams	0.001	0.05	10.00	0.001	0.11	10.00	0.001	0.30	10.00
$A_3(m^2)$	Transverse deck beams	0.001	0.05	10.00	0.001	0.10	10.00	0.001	0.10	10.00
$A_7(m^2)$	Deck truss diagonals				0.001	0.80	10.00	0.001	0.20	10.00
$A_4(m^2)$	Arch ribs	0.001	0.10	20.00	0.001	0.80	20.00	0.001	2.00	20.00
$A_5(m^2)$	Deck Hangers	0.001	0.05	10.00	0.001	0.10	10.00	0.001	0.10	10.00
$A_6(m^2)$	Lateral braces	0.001	0.01	10.00	0.001	0.05	10.00	0.001	0.05	10.00
γ_z	Span-to-rise ratio of arch	3.0	5.0	20.0	3.0	5.0	20.0	3.0	5.0	20.0
γ_y	Span-to-inward lean of arch	6.0	10.0	300.0	21.0	80.0	1000.0	21.0	80.0	1000.0

Trussed arch bridges

Eqs. 3.4 and 3.5 define the geometry of the **trussed arch** bridge models (Fig. 3.5).

$$z_{\text{inner}}(x) = \frac{(x - h_{in})^2}{4h_{in} + \frac{-L^2}{16h_{in}}} \tag{3.4}$$

where x and z are the x- and z-coordinates of points along the inner parabolic chord of the arch, which has crown height h_{in} and span L. The geometry of the outer (upper) chord of the arches varies polynomially according to Eq. 3.5.

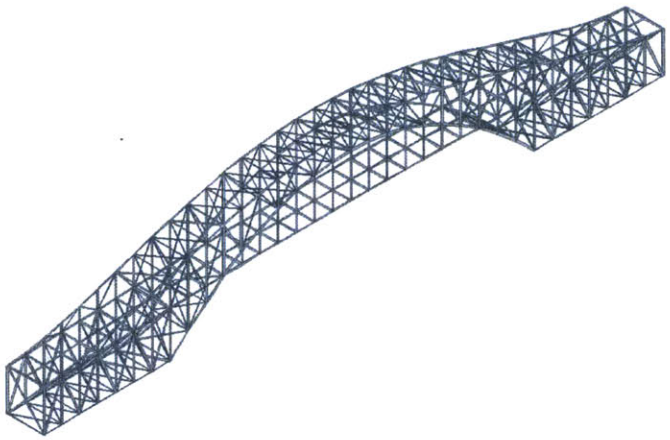
$$z_{\text{outer}} = C_0 + C_1 x + C_2 x^2 + C_3 x^3 + C_4 x^4 \tag{3.5}$$

where x and z are the x- and z-coordinates of points along the outer parabolic chord of the arch. The constants C_i are determined by fitting a fourth-order polynomial through five known points: the ends of the curved outer chord and the points at quarter, half, and three-quarters of the distance along the chord's span. The height of the inner chord at midspan, h_{in}, and the vertical distance between the inner and outer chords at midspan—

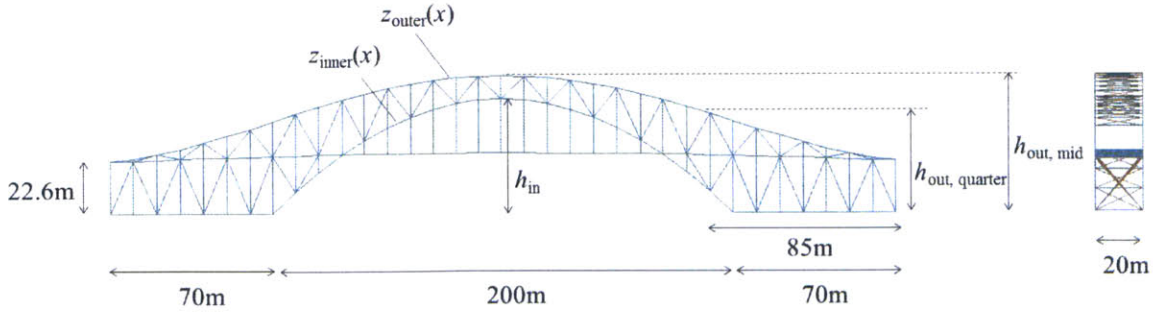
the truss's depth, d_{mid}—are the two geometric design variables used in optimization. The remaining heights used to fit Eq. 3.5 are given by:

$$\begin{aligned} h_{out,mid} &= h_{in} + d_{mid} \\ h_{out,quarter} &= h_{out,mid} - 0.3h_{in} \end{aligned} \tag{3.6}$$

Table 3.5 shows the starting values and bounds for the 200 m and 300 m trussed arches. The models' highly interconnected nature and their multiple elements require more sizing variables than the other bridge types.



(a) Isometric projection



(b) Side elevations

Figure 3.5 – Structural model of the 200 m steel trussed arch bridge. The design variables h_{in} and $h_{out,mid}$ control the geometry, and nine additional sizing variables determine the superstructure's element sizes. (Table 3.5 shows the design variables' starting values and bounds.)

Table 3.5 – Starting values and bounds for the trussed arch models' design variables

		200 m			300 m		
		LB	Value	UB	LB	Value	UB
$A_1(\mathrm{m}^2)$	Outer longitudinal deck beams	0.001	0.05	10.00	0.00	0.05	10.00
$A_2(\mathrm{m}^2)$	Inner longitudinal deck beams	0.001	0.05	10.00	0.00	0.05	10.00
$A_3(\mathrm{m}^2)$	Transverse deck beams	0.001	0.15	10.00	0.00	0.80	10.00
$A_4(\mathrm{m}^2)$	Inner arch ribs	0.001	0.10	10.00	0.00	0.25	10.00
$A_5(\mathrm{m}^2)$	Outer arch ribs	0.001	0.05	10.00	0.00	0.05	10.00
$A_6(\mathrm{m}^2)$	Truss verticals	0.001	0.10	10.00	0.00	0.50	10.00
$A_7(\mathrm{m}^2)$	Truss diagonal below deck	0.001	0.01	10.00	0.00	0.10	10.00
$A_8(\mathrm{m}^2)$	Truss diagonal above deck	0.001	0.10	10.00	0.00	0.50	10.00
$A_9(\mathrm{m}^2)$	Truss bottom chord	0.001	0.01	10.00	0.00	0.10	10.00
$A_{10}(\mathrm{m}^2)$	Lateral trasverse braces	0.001	0.01	10.00	0.00	0.01	10.00
$A_{11}(\mathrm{m}^2)$	Deck tension hangers	0.001	0.05	10.00	0.00	0.10	10.00
$A_{12}(\mathrm{m}^2)$	Lateral diagonal braces	0.001	0.01	10.00	0.00	0.01	10.00
$h_{in}(\mathrm{m})$	Rise of inner arch	20.0	5.0	100.0	30.0	90.0	120.0
$d_{mid}(\mathrm{m})$	Midspan vertical arch spacing	5.0	10.0	30.0	5.0	10.0	20.0

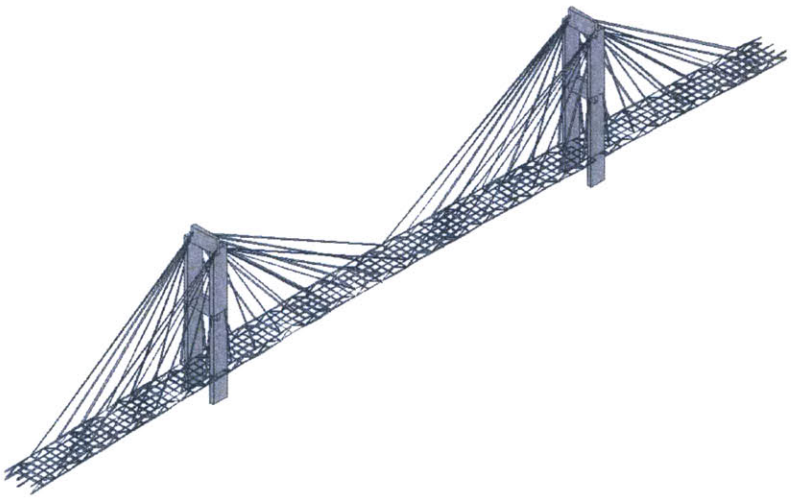
Cable-stayed bridges

Cable-stayed bridges, although occasionally encountered as aesthetically-pleasing short-span footbridges, are most commonly used in long-span traffic-carrying situations. In this work, the three cable-stayed bridges—spanning 300 m, 500 m, and 800 m—are configured in a fan pattern, where all the cables run from the deck to the top of the towers.

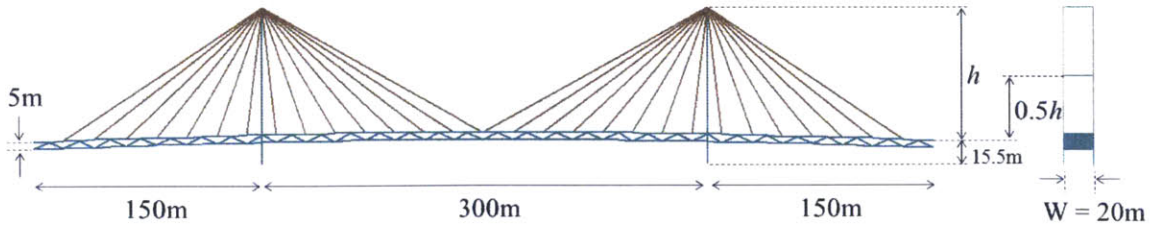
The height of the towers, h, is the only geometric design variable. In addition to the deck sizing variables, two additional parameters, A_5 and A_6, determine the size of the cables and the towers. The cables elements do not use the sizing procedure from §3.1.1.

Instead, they are assumed to have no bending rigidity, with I_{maj}, I_{min}, and J set to zero. The only other elements in the study that use this sizing procedure are the suspension bridges' primary cables and hangers.

Fig. 3.8 shows the 300 m cable-stayed model, and Table 3.6 shows the design variables' starting values and bounds. The analysis of these bridges accounts for geometric nonlinearity, later described in §3.3.1.



(a) Isometric projection



(b) Side elevations

Figure 3.6 – In geometry optimization problems, the 300 m cable-stayed bridge model's geometry is controlled by h, the height of its towers. In addition to the deck element sizing variables, two design variables controlling the cross-sectional area of the tower elements (A_5) and the cables (A_6) are used.

Table 3.6 – Starting values and bounds for the three cable-stayed bridge models' design variables

		300 m			500 m			800 m		
		LB	Value	UB	LB	Value	UB	LB	Value	UB
$A_1(\mathrm{m}^2)$	Outer long. deck beams	0.03	0.04	10.00	0.05	0.06	20.00	0.10	0.10	20.00
$A_2(\mathrm{m}^2)$	Inner long. deck beams	0.03	0.03	10.00	0.05	0.06	20.00	0.10	0.10	20.00
$A_3(\mathrm{m}^2)$	Transverse deck beams	0.03	0.06	10.00	0.05	0.06	20.00	0.10	0.10	20.00
$A_6(\mathrm{m}^2)$	Deck truss diagonals	0.02	0.04	10.00	0.01	0.03	20.00	0.03	0.03	20.00
$A_4(\mathrm{m}^2)$	Towers	5.00	7.00	100.00	5.00	7.00	100.00	5.00	8.00	100.00
$A_5(\mathrm{m}^2)$	Cables	0.03	0.03	10.00	0.05	0.06	20.00	0.10	0.10	20.00
$h(\mathrm{m})$	Tower height	30.0	90.0	150.0	50.0	150.0	250.0	160.0	240.0	280.0

Girder bridges

The 50 m **girder** bridge (Fig. 3.7) is basically a cambered deck without any additional superstructure. Its design is therefore determined by the three deck sizing variables—A_1, A_2, and A_3 (Table 3.7)—and its geometry does not vary.

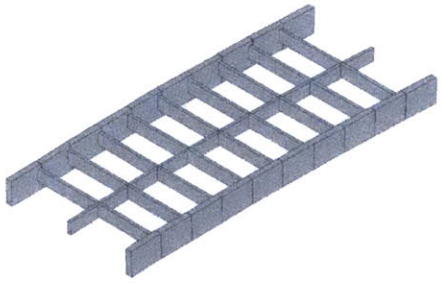


Figure 3.7 – The 50 m girder bridge model is a deck without additional superstructure. Accordingly, problems based on this model only use the three sizing variables that control deck design.

Table 3.7 – Starting values and bounds for the girder bridge model's design variables

		50 m		
		LB	Value	UB
$A_1(\mathrm{m}^2)$	Outer longitudinal deck beams	0.001	2.80	10.00
$A_2(\mathrm{m}^2)$	Inner longitudinal deck beams	0.001	1.00	10.00
$A_3(\mathrm{m}^2)$	Transverse deck beams	0.001	1.00	10.00

Suspension bridges

The two **suspension bridges**, spanning 500 m and 800 m, are modeled in the same way as the suspension bridge in Chapter 1's preliminary study.

The cable geometry varies according to Eq. 3.7, and the presence of the back spans and the bridge towers is represented by fully restraining the nodes at the top of the cables. The analysis includes the effects of geometric nonlinearity, as discussed in §3.3.1.

$$z(x,p) = \frac{(x-\frac{L}{2})^2}{4p} + 3 \tag{3.7}$$

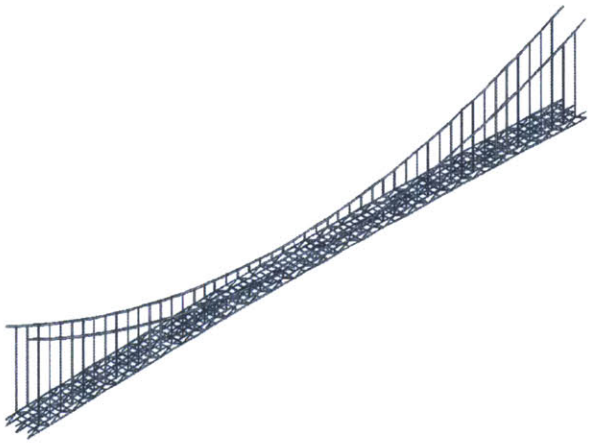
where x and z are the x- and z-coordinates the suspension cables, L is the bridge span,

Table 3.8 – Starting values and bounds for the 500 m and 800 m suspension bridge models' design variables

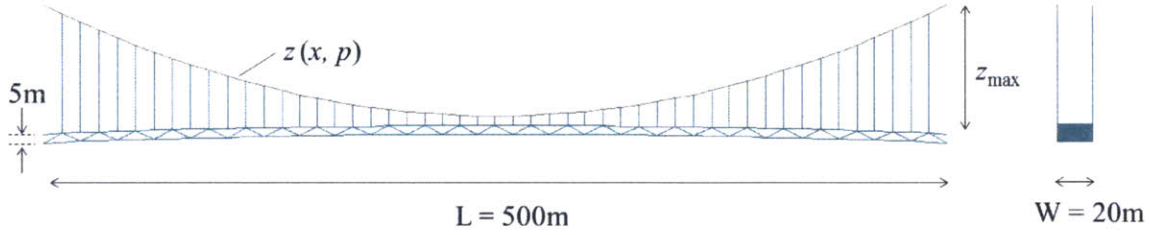
		500 m			800 m		
		LB	Value	UB	LB	Value	UB
$A_1(\mathrm{m}^2)$	Outer longitudinal deck beams	0.001	0.10	10.00	0.001	0.10	10.00
$A_2(\mathrm{m}^2)$	Inner longitudinal deck beams	0.001	0.10	10.00	0.001	0.10	10.00
$A_3(\mathrm{m}^2)$	Transverse deck beams	0.001	0.10	10.00	0.001	0.10	10.00
$A_6(\mathrm{m}^2)$	Deck truss diagonals	0.001	0.40	10.00	0.001	0.40	10.00
$A_4(\mathrm{m}^2)$	Primary cables	0.001	0.10	10.00	0.001	0.10	10.00
$A_5(\mathrm{m}^2)$	Hangers	0.001	0.10	10.00	0.001	0.10	10.00
$p(\mathrm{m})$	Parametric parameter. See Eq.3.7.	100.0	250.0	1000.0	100.0	400.0	1000.0

and p is a geometric design variable that controls the depth of the parabola.

Table 3.8 gives the starting values and design variables for the 500 m and 800 m suspension bridge models, the former of which is shown in Fig. 3.8.



(a) Isometric projection



(b) Side elevations

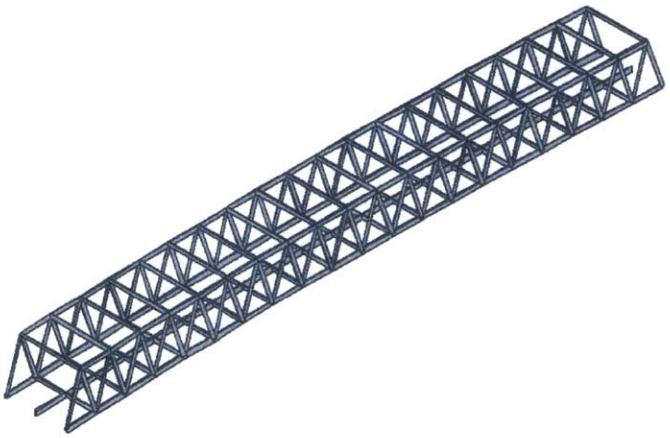
Figure 3.8 – The 500 m suspension bridge, with cable geometry defined by Eq. 3.7, is analyzed accounting for geometric nonlinearities.

Warren truss bridges

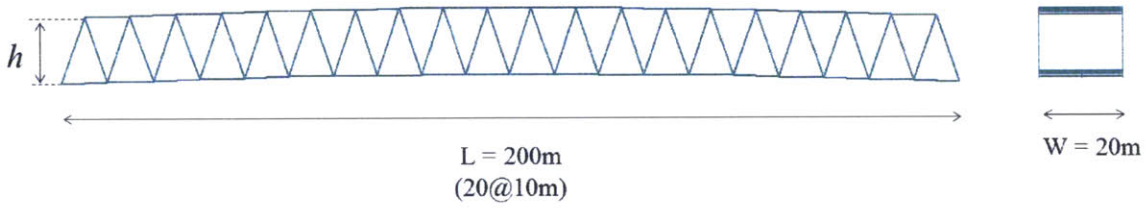
Bridges of the final type—**Warren truss** bridges—have a top and bottom truss chord, both of which, like the road decks, have a parabolic precamber with a maximum span-to-rise ratio of 100 : 1. The chords are linked by diagonal web elements (Fig. 3.9).

The distance between the top and bottom chords, h, is only the geometric design variable. Additional sizing variables—A_4, A_5, and A_6—determine the size of the trusses' top

chords, the struts that brace those chords, and the diagonal elements that connect them to the deck (Table 3.9).



(a) Isometric projection



(b) Side elevations

Figure 3.9 – Views of the 200 m Warren truss bridge show the single geometric design variable h. Additional sizing variables A_4, A_5, and A_6 control the size of top truss chords, the struts that brace them, and the diagonal connectors.

3.1.3 Range of optimization problems: experimental design

The fourteen structures in §3.1.2 form the basis of 474 optimization problems. Different problems are formulated for a given structure by varying the objective function, the number and type of constraints, the nature of the design variables and, in the case of sizing problems (where geometry remains constant), the height-to-span ratio of the starting design.

A number of other features distinguish problems, including the degree of constraint and other mathematical characteristics of the design space. These secondary problem

Table 3.9 – Starting values and bounds for the three Warren truss models' design variables

		50 m			200 m			300 m		
		LB	Value	UB	LB	Value	UB	LB	Value	UB
$A_1(\mathrm{m}^2)$	Outer long. deck beams	0.001	0.02	10.00	0.001	0.50	10.00	0.001	0.80	10.00
$A_2(\mathrm{m}^2)$	Inner long. deck beams	0.001	0.02	10.00	0.001	0.50	10.00	0.001	0.50	10.00
$A_3(\mathrm{m}^2)$	Transverse deck beams	0.001	0.02	10.00	0.001	0.20	10.00	0.001	0.20	10.00
$A_4(\mathrm{m}^2)$	Truss top chords	0.001	0.02	10.00	0.001	0.40	10.00	0.001	0.80	10.00
$A_5(\mathrm{m}^2)$	Lateral truss bracing	0.001	0.02	10.00	0.001	0.20	10.00	0.001	0.20	10.00
$A_6(\mathrm{m}^2)$	Truss diagonals	0.001	0.02	10.00	0.001	0.20	10.00	0.001	0.20	10.00
$h(\mathrm{m})$	Truss height	5.0	10.0	20.0	5.0	15.0	40.0	8.0	20.0	60.0

features—so-called because any variation in them from problem to problem is a side effect of intentional variation in the primary characteristics—are also used to characterize problems later in the dissertation.

The experimental design is full factorial, using every feasible combination of the problem features' levels. This is enabled by having a discrete and manageable set of levels for each feature. For example, bridge span—a feature which, in reality, varies continuously—is restricted to the discrete set of values in Table 3.3.

The primary problem features and their selected levels are:

- **Structural type:** Each problem is based on one of the six structural types described in the preceding section.
- **Span:** A few appropriate spans for each structural type are chosen (Table 3.3).
- **Problem type:** The study covers two of the three main types of optimization problems defined in §2.1: sizing and geometry optimization. Topology problems are beyond the scope of this study, but are an interesting area for future investigation.
  - *Sizing:* In sizing problems, algorithms vary only the size of the structural elements. The goal is to distribute material, and overall geometry remains constant.
  - *Simultaneous geometry and sizing:* In these problems, design variables that control structural models' geometry are included, and algorithms vary both

geometry and element sizes in search of the best solution. Girders are the only bridges for which geometry never varies.

- **Objective function**
  - *Minimum mass:* When minimizing mass, the goal is to find the lightest feasible design. The mass model accounts only for the mass of the superstructure and the deck beams; it does not include the mass of the bridge decking and roadway finishes.
  - *Maximum stiffness:* The stiffness objective function is set according to Eq. 3.2 in §3.1.1.
- **Constraints:** In each problem, some aspect of structural behavior is constrained.
  - *Displacements:* For displacement-constrained problems, the absolute displacement of all nodes along a bridge's deck due to unfactored live load (applied to either all or half of the roadway) must not exceed $\frac{1}{800}$ of the bridge's span.
  - *Stresses:* When stress constraints are included, the maximum stress in every structural element is checked against the material's yield stress (Table 3.2). These constraints are evaluated when combined, factored live and dead loads (Table 3.1)..
  - *Mass:* Although displacement and stress can be constrained for any problem, it is only feasible to constrain mass when it has not already been chosen as the objective function. In these cases—i.e., when maximum stiffness is the objective—an appropriate upper-bound constraint on overall mass is set.
- **Height-to-span ratio:** For problems where the geometry does not vary, characteristics of the initial geometry become suitable problem features, and the height-to-span ratio is the most intuitive to use. We establish upper and lower bounds on height-to-span for each bridge (shown in Tables 3.4 through 3.9), and generate optimization problems at each of five evenly-spaced values in that range.

As Appendix B.3 explains in detail, certain levels of the primary problem features cannot be combined (e.g., mass cannot be constrained when it is also the objective), leading to a 474-problem experimental design that falls just short of full factorial.

To generate the dataset on which the dissertation's results are based, eight different optimization algorithms solve each of the 474 problems, resulting in 3792 problem-algorithm combinations. The following section describes those algorithms.

3.2 Optimization algorithms

The design problems are solved using eight algorithms from the NLopt non-linear optimization library [Johnson, 2012] (Table 3.10). The algorithms represent the major classes of derivative-free[2] nonlinear solvers, including linear and quadratic approximations, simplex methods, directional search, and stochastic evolutionary methods. Each algorithm has a unique approach to progressing through the design space.

They are categorized into *local algorithms*, which are designed to converge on a region of the design space where no improvement can be made by slight design perturbations in any direction, and *global algorithms*, which perform a broader search of the design space to locate the overall global optimum. Some algorithms naturally solve only unconstrained optimization problems; in these cases the objective and constraint functions are combined in a single *Augmented Lagrangian* formulation, allowing all algorithms to address all problems [Conn et al., 1991].

Table 3.10 – The same eight algorithms used in Chapter 1 solve each optimization problem.

Local	Constrained Optimization by Linear Approximation–*COBYLA* [Powell, 1994]
	Bounded Optimization by Quadratic Approximation–*BOBYQA* [Powell, 2009]
	Nelder-Mead Simplex—*NEL-MEAD* [Nelder and Mead, 1965]
	Subplex—*SUBPLEX* [Rowan, 1990]
	Principal Axis Method—*PR-AXIS* [Brent, 2002]
Global	Dividing Rectangles Method—*DIRECT* [Jones et al., 1993]
	Controlled Random Search—*CRS* [Kaelo and Ali, 2006]
	Improved Stochastic Ranking Evolution Strategy—*ISRES* [Runarsson and Yao, 2005]

3.2.1 Description of algorithms

The remainder of the dissertation refers to each of the algorithms using either the abbreviation or the acronym of their full name specified in the table above.

[2]Despite the presence of gradient-based algorithms in the structural optimization literature, we omit them because of the computational expense of approximating gradients using commercial analysis software. These methods could of course be evaluated in future studies, especially if open-source structural analysis software from which gradients can be directly extracted is available.

Constrained Optimization by Linear Approximation—*COBYLA*

COBYLA is a derivative-free algorithm which constructs successive linear approximations of the objective and constraint functions via a simplex of size $n+1$ (where n is the number of design variables) and optimizes over these approximations in a trust region [Powell, 1994].

COBYLA converges quickly for convex or near-convex problems. Although its approximation strategy makes it reasonably insensitive to local noise in the objective and constraint functions, it may become stuck in a local optimum.

Bounded Optimization by Quadratic Approximation—*BOBYQA*

The BOBYQA algorithm performs bound-constrained optimization on an iteratively constructed quadratic approximation of the objective function [Powell, 2009]. Its performance should be relatively similar to COBYLA's, although quadratic approximation may be problematic when the design space is far from being twice-differentiable.

Nelder-Mead Simplex—*NEL-MEAD*

NEL-MEAD constructs a simplex of $n + 1$ design points and updates that simplex by reflecting, expanding, or contracting its vertices [Nelder and Mead, 1965]. The NLopt implementation of the algorithm accounts for bound constraints on the design variables using the method described in Box [1965].

We expect relatively strong performance on noisy objective and constraint functions. Although the algorithm may become stuck in local optima, the strategy of simplex reflection and expansion increases the likelihood of escaping local optima and reaching the global optimum.

Subplex—*SUBPLEX*

This subspace-searching simplex algorithm, like the Nelder-Mead method that it generalizes, works well with noisy objective functions [Rowan, 1990]. By searching subspaces, the number of function evaluations required for convergence usually increases linearly with the problem size.

For most problems, Subplex combines the advantages of Nelder-Mead with improved efficiency, and should converge on a similar answer with fewer simulation calls.

Principal Axis Method—*PR-AXIS*

PR-AXIS is a refinement of Powells well-known *Conjugate Gradient Descent* method [Powell, 1964], which minimizes a function by a series of binomial searches along prescribed vectors [Brent, 2002].

It should perform well on complex objective functions, although it is likely to struggle when faced with many local optima in the search space.

Dividing Rectangles Method—*DIRECT*

This deterministic global optimization algorithm systematically divides a hyper-cube representation of the search space into progressively smaller hyper-rectangles, eventually converging on the global optimum [Jones et al., 1993].

It should find the global optimum, but is likely to require many simulation calls. This may prove problematic on the cable-stayed and suspension bridges, whose nonlinear analysis is computationally expensive.

Controlled Random Search—*CRS*

The CRS family of stochastic algorithms are similar to Genetic Algorithms (GAs). They start with a randomly-generated population of solutions and evolve these solutions using heuristics, though these heuristics are closer to the spirit of Nelder-Mead that the mutation and crossover operations of traditional GAs [Kaelo and Ali, 2006].

CRS should find the global optimum regardless of the mathematical characteristics of the search space. Its population-based approach, however, requires many objective and constraint evaluations and many calls to the analysis software.

Improved Stochastic Ranking Evolution Strategy—*ISRES*

This stochastic algorithm generates a population of solutions using the *NLopt* library's default population size of $20(n + 1)$ [Runarsson and Yao, 2005]. The algorithm evolves these solutions using a combination of a GA-style mutation rule and differential variation, and ranks solutions in the population using the stochastic scheme proposed by Runarsson and Yao. It should perform similarly to CRS.

Algorithm convergence criteria

The local algorithms terminate when the objective function converges within a relative tolerance of 1×10^{-4}. This convergence criterion makes little sense for the three global algorithms; we instead set these to terminate after six hours. Although it could be argued that, for simulation-based problems, six hours is an unfairly-short time constraint to impose on such algorithms, it is a realistic upper bound to the amount of time for which a practicing engineer could reasonably await a solution.

3.2.2 Evaluating algorithm performance

As noted in §1.2, there are many ways to evaluate algorithm performance. Furthermore, the choice of performance measure can significantly influence perceptions of relative performance levels. The computational setup (described next in §3.3 and §3.4) measures and records the following performance data:

- **Final design evaluation:** The value of the final objective function, and the filepath to the final version of the structural model to enable more extensive evaluation.
- **Counters:** The software records integer counts of the number of calls made to the analysis software, to the objective function, and to the constraint functions.
- **Design vector values:** The value of the design vector $\vec{x}$ when the algorithm terminates, and all intermediate values of $\vec{x}$, to allow tracking of how soon the algorithm approaches its final answer.
- **Clock time:** The elapsed time between starting the algorithm (after all initialization takes place) and convergence.

Normalizing performance measures

The stated goals of this work necessitate comparisons of algorithm performance across different optimization problems, which have different degrees of difficulty and different 'best' answers. A relatively well-designed and light 500 m suspension bridge, for instance, may weigh twice as much as a similarly well-designed 300 m suspension bridge, making direct comparison between the quality of an algorithm's solution for both problems inappropriate. Or a stiffness objective function for a bridge could be considerably more complex than the same bridge's mass objective function, requiring an algorithm to make more function calls and making direct comparison of computational efficiency similarly inappropriate.

To enable inter-problem comparison of algorithm performance, we scale an algorithm's performance on a problem between the best and worst observed performances among all algorithms on that problem, resulting in the normalized performance measures $\bar{y}$ (normalized objective value) and $\bar{c}$ (normalized computational cost); $0 \leq \bar{y}, \bar{c} \leq 1$ (Eq. 3.8).

$$\begin{aligned}
\bar{y}_{i,j} &= \frac{y_{\max,j} - y_{i,j}}{y_{\max,j} - y_{\min,j}} && i \in \{\text{All algorithms}\}; \quad j = 1..d \\
\bar{c}_{l,j} &= \frac{c_{\max,l,j} - c_{l,j}}{c_{\max,l,j} - c_{\min,l,j}} && l \in \{\text{Local algorithms}\}; \quad j = 1..d \\
\bar{c}_{g,j} &= \frac{c_{\max,g,j} - c_{g,j}}{c_{\max,g,j} - c_{\min,g,j}} && g \in \{\text{Global algorithms}\}; \quad j = 1..d
\end{aligned} \tag{3.8}$$

where $y_{i,j}$ is the observed performance of algorithm i on problem j, and $y_{\min,j}$ and $y_{\max,j}$ are the best and worst observed performances among all algorithms on the same problem j. The computational cost measure, c, is normalized in the same way, except that local and global algorithms' measures are normalized among only algorithms of the same type. This accounts for the very different numbers of analysis software calls—often several orders of magnitude—made by local and global algorithms. The normalized measures all lie in the range $[0, 1]$, with higher values indicating better performance.

3.3 Software

Running the optimization problems described in the previous section requires three main software components: a structural analysis engine to evaluate design performance, an optimization library which implements the chosen algorithms, and a data management system to store and manipulate the extensive results. This section describes all three.

3.3.1 Structural analysis software

Oasys's General Structural Analysis (GSA)

All structural modeling, analysis, and visualization is performed using version 8.6 of Oasys's General Structural Analysis (GSA) Suite [Oasys, 2012]. Despite individual variations across some implementation details, the models all conform to the specifications of §3.1.1, including loading patterns and material properties (Tables 3.1 and 3.2).

Four of the six bridge types—basket-handle arches, trussed arches, girders, and Warren trusses—are analyzed linearly using *GSS*, GSA's *Direct Stiffness Method* solver. We use a sparse matrix solver for the resulting system of linear equations by setting GSS's *Matrix Solver* option to "Sparse Direct", which uses the *Approximate Minimum Degree* algorithm to order degrees of freedom and the *LDL Sparse* algorithm for matrix factorizing [Davis, 2005; Amestoy et al., 2004].

The analysis of the remaining bridge types—cable-stayed and suspension—accounts for geometric nonlinearity using GSA's dynamic relaxation solver, *GsRelax*. A dynamic relaxation solver applies loads, causing an initial set of displacements, and iterates until all internal forces are in equilibrium. GsRelax uses viscous damping material damping, and checks force and moment residuals for convergence to within 0.001% after every ten analysis cycles, up to a maximum of 100,000 cycles.

Custom wrapper around GSA

GSA provides a *Common Object Model* (COM) *Application Programming Interface* (API) which allows model manipulation and results evaluation by passing text strings to and from the program[3].

To integrate GSA with the the rest of the software, a wrapper around this COM API provides a set of methods to generate and manipulate models, set up and run structural analyses, and retrieve results. The wrapper is written in the C# language to enable easy integration of GSA in software projects developed with Microsoft's *.NET* environment, and is available online at `https://github.com/roryclune/GSADotNet` [Microsoft, 2013b].

The wrapper methods are called within a .NET project using a set of user-specified arguments, which the methods use to generate and interpret text strings sent to and received from the COM API. These methods are the key to integrating GSA with the rest of the software used in this work.

Having established a means of modeling and analyzing structures programmatically, the next requirement in building the software infrastructure is a library of optimization algorithms and a means of integrating them in the .NET environment.

[3]The syntax of these strings and the functionality they expose are described in GSA 8.6's manual, available online at `http://www.oasys-software.com/media/Manuals/Latest_Manuals/gsa8.6_manual.pdf`

3.3.2 Optimization library

NLopt nonlinear optimization library

We use the implementation of the eight optimization algorithms from version 2.2.3 of the *NLopt* nonlinear optimization library, available for public use under the *GNU Lesser General Public License* (Table 3.10) [Johnson, 2012].

The library is originally written in C code for computers running UNIX operating systems, but its developer also provides a dynamic linked library (DLL) for use on *Windows*®, wrapping the original functions in a set of C++ functions. This DLL, however, is written for use in unmanaged code environments, whereas the .NET environment used here requires managed code libraries. (See Heege [2007] for a discussion of managed and unmanaged code.)

An additional wrapper around the original C++ wrapper enables the use of the unmanaged NLopt library. It is written, as part of this work, using C++/CLI—a language designed by Microsoft for bridging the gap between managed and unmanaged environments [ECMA International, 2013].

C++/CLI wrapper around NLopt

In terms of functionality, the C++/CLI wrapper around the original C++ wrapper simply allows the latter's methods to be called in a managed code environment. The principal challenges are the casting between managed and unmanaged data types and the tracking of memory locations on the unmanaged heap.

The wrapper, along with instructions for its use in .NET, is available online at `https://github.com/roryclune/NLoptDotNet`.

3.3.3 Data management

Solving so many optimization problems with multiple algorithms generates large volumes of data, which are not easily managed ad-hoc. A unique contribution of this dissertation is its comprehensive strategy for representing and queuing problems, for storing results and algorithm performance measures, and for enabling easy data manipulation and interpretation during the results chapters.

The relational data model is an important contribution of this research (Fig. 3.10). It embodies a strategy for organizing and extracting information from multiple structural

optimization problems, which makes it useful both for future researchers in this field and for developers of optimization systems for use in the structural design industry. The schema also provides a single clear representation of the information that can later be used to develop an algorithm selection system. This importance motivates a full description of its entities and relationships, and of their origins in the underlying problem.

The fully-normalized model is implemented in the *SQL Server® 2008 R2* relational database management system [Microsoft Technet, 2013]. (See Kent [1983] for a discussion of the five normal database forms.) The *Linq-to-SQL* extension to the C# programming language is used throughout to retrieve queued optimization problems and write their results [Microsoft, 2013a].

In the model, each of the 474 optimization problems from §3.1.3 is represented by a unique *Problem Instance* entity. Each *Problem Instance* has a *Structure* parent entity; each of the fourteen *Structures* (§3.1.2) can be an attribute of multiple *Problem Instances*, reflecting the fact that many optimization problems are formulated for each bridge structure. *Structure* is, in turn, a super-type of many different types of structures and bridges, each of which has its own descriptive attributes.

Problem Instances are further described by their many-to-many relationships with *Problem Type* and *Objective* entities, which describe the nature of the design variables and the target of the objective function.

The attributes of these entities collectively store the characteristic problem features enumerated in §3.1.3.

The representation of optimization algorithms centers on the *Solver* entity, which can have many *Algorithms* as attributes. This reflects the fact that algorithms can be hybrids, usually of two algorithms. In this study, for example, the Augmented Lagrangian formulation—whose usefulness in solving constrained problems with algorithms intended for unconstrained problems is outlined in §3.2—is treated as a hybrid of an Augmented Lagrangian algorithm, which brings the constraints into the objective function, and a secondary algorithm which solves the resulting unconstrained problem.

A *Solver* also has a number of *Parameter* entities as attributes; these store the values of termination criteria and any relevant tuning parameters, such as the initial population size for a population-based stochastic algorithm.

Every algorithm is used to solve every problem once, and each of these exhaustive algorithm-problem pairings is represented by a unique *Solver-Problem Instance* entity. The attributes of this entity—apart from its *Design Variable* children, which record the start, end, and intermediate design points—store the algorithm performance data. The *Solver-Problem Instance* entities also store the location of relevant structural model files.

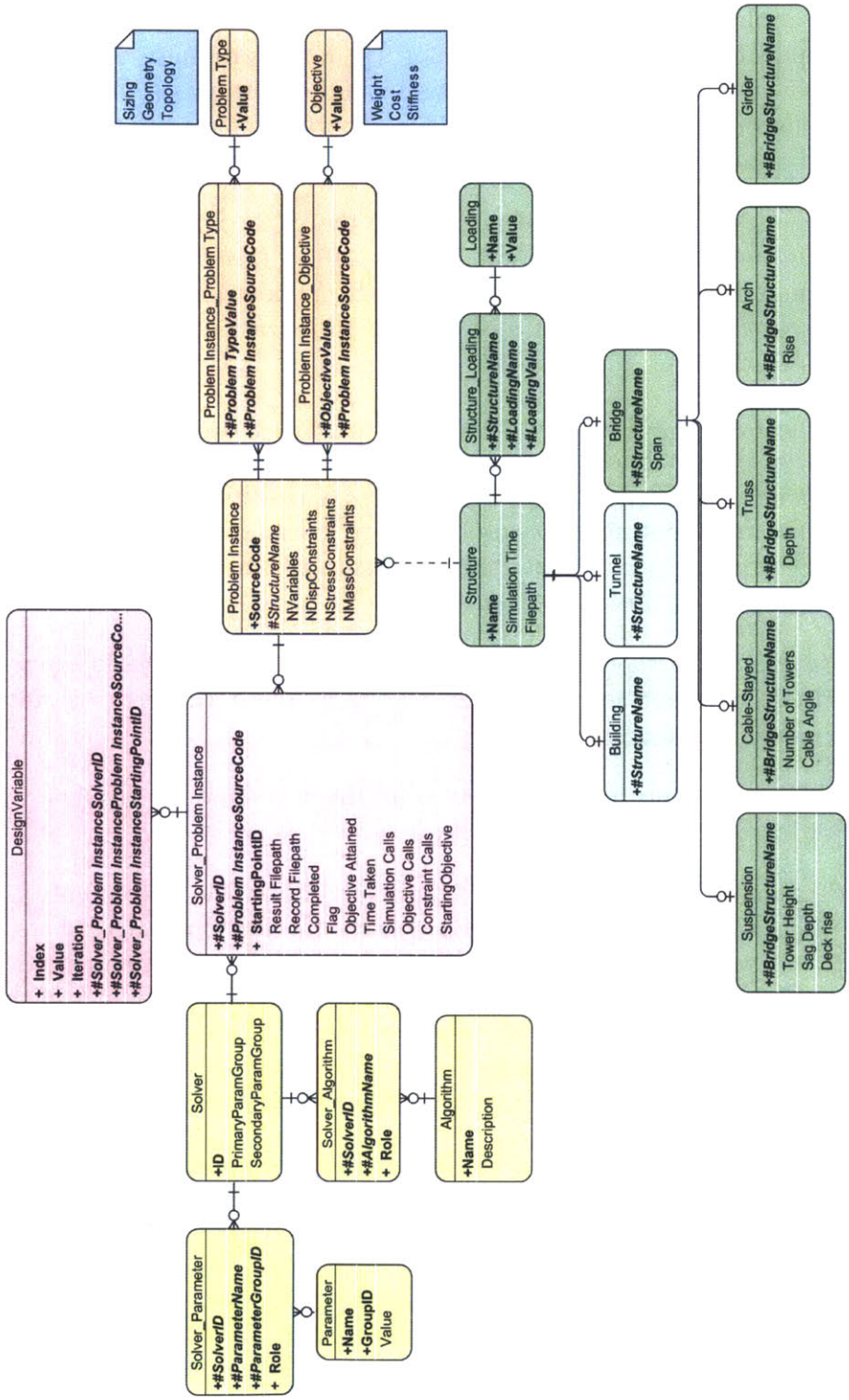


Figure 3.10 – The data model, shown here in a UML entity-relationship diagram, embodies a strategy for organizing and extracting information from multiple structural optimization problems.

3.4 Hardware and operating system

All computation takes place on a server running the 64-bit version of *Windows Server® 2008 R2 Enterprise* (upgraded to *Service Pack* 1). The server's *Tyan® S7025AGM2NR* motherboard is fitted with two *Intel® Xeon® E5520* 2.27 GHz quad-core CPUs, each having eight threads, 8 MB of *Intel® Smart Cache*, and a bus/core ratio of 17.

Each CPU is connected to four 4 GB *DDR3-SDRAM* modules, for a total installed memory of 32 GB. File are written to and read from a 640 GB *Western Digital® WD6400AAKS* Serial ATA hard drive with 16 MB cache, a 7200 RPM rotation speed, and a 3.0 Gb/s interface speed.

Summary

This chapter presents the computational method used to generate optimization data, including the optimization problems, the algorithms used to solve them, and the computational infrastructure that runs problem-algorithm pairs and manages data. The data generated with the presented method is used in the rest of the dissertation to understand and address the algorithm selection problem identified in Chapters 1 and 2.

This chapter introduced a number of novel contributions to the field, including a continuous approximation to discrete element sizing, a specification of a stiffness objective function, the .NET wrappers around the GSA analysis software and NLopt optimization library, and a data model to represent problems and algorithms and to record data on their performance.

The next chapter evaluates the generated results an engineering point of view. Its purpose—apart from helping us to better understand the generated data set and the range of solutions produced by different algorithms—is to demonstrate the engineering rationale and feasibility of the solutions. This roots our later work on algorithm performance trends and automated selection techniques (Chapters 5 and 6) in the world of structural engineering, strengthening the practical applicability of the research.

Chapter 4

Results: engineering verification of data

It is tempting, given the dissertation's goal of addressing algorithm selection, to immediately begin using the data generated in Chapter 3 to explore the relationship between algorithms' performance and features of the problems they solve. The strength of conclusions derived from such an exploration, however, depends on the integrity of the optimization outcomes, and investigating the various solutions produced by algorithms furthers our understanding of the problem. These two considerations warrant a close inspection of the data from an engineering perspective.

This chapter, therefore, examines the results generated by Chapter 3's computational method, verifying the optimization results without regard to which algorithms produced them. A demonstration of the results' validity as solutions to real structural engineering problems would enhance the applicability of the dissertation's findings to real-world design scenarios.

To achieve this, §4.1 looks at overall trends across the dataset; §4.2 then selects and evaluates representative solutions for each bridge type. The data contain hundreds of designs for each bridge structure[1]. This latter section therefore uses clustering to produce a manageable, representative subset of designs which can feasibly be evaluated one by one.

[1]The experimental design—§3.1– generates tens of optimization problems from each of fourteen bridge structures, all of which are solved by eight optimization algorithms.

4.1 Overall structural performance

A useful quantity which allows engineers to quickly evaluate a bridge's structural efficiency is the mass per unit area of road deck. This metric enables comparison across all bridge types and against existing, real-world structures.

Fig. 4.1 plots mass per unit deck area against span for the fourteen bridge models. Since the purpose of this plot is to broadly visualize material use, it uses only a single value of mass per unit area to represent the hundreds of available designs for each model. This single value is the mass of the lightest design produced for a mass minimization problem with LRFD-compliant constraints on all element stresses and nodal displacements under multiple load cases (§3.1.1). This ensures the chosen designs are as close to reality as possible.

Before plotting, the data are modified to account for unmodeled aspects of the structures which would be present in reality (see §3.1.2 for modeling details).

- To account for the presence of road decks, represented in the models by an imposed dead load of $3.80\,\mathrm{kN/m^2}$, each model's mass per unit area is increased by $\frac{3.80\times1000}{9.8} = 388\,\mathrm{kg/m^2}$, the mass per unit area required to generate this dead load (assuming acceleration due to gravity of $9.8\,\mathrm{m/s^2}$).

- To include the mass of the suspension bridges' towers, their required cross-sectional area is computed by dividing a force equal to half the weight of the main span (which a tower would have supported), plus the live and dead load supported by that span, by the yield stress of steel ($300\,\mathrm{N/m^2}$). This area is then multiplied by the tower height (which is dependent on the main cables' sag) and by the density of steel ($7850\,\mathrm{kg/m^3}$) to estimate the towers' mass.

- The two back-spans of the suspension bridge—the sections which extend from the towers to the outer supports—are not explicitly modeled. To compensate, the suspension bridges' mass is multiplied by 1.5. Although a crude approximation, this is based on a series of reasonable assumptions: that each back-span has a length equal to a quarter of the main span, that back-spans have the same deck configuration as the main span, and that the cables supporting each back span weigh a quarter as much as each main cable.

- In response to large local displacements and resulting numerical instabilities observed during the geometrically nonlinear structural analyses of cable-stayed bridges in the study's early stages, we impose artificially-high lower bounds on the deck beam sections in the cable-stayed formulations (Table 3.6). This leads to optimization-generated designs which are unnecessarily heavy, but which can be

manually modified without unduly affecting overall displacements or stresses. Before plotting, we remove all excess material from the cable-stayed decks; this mostly consists of reducing the size of lateral bracing elements, which has little effect on the overall behavior.

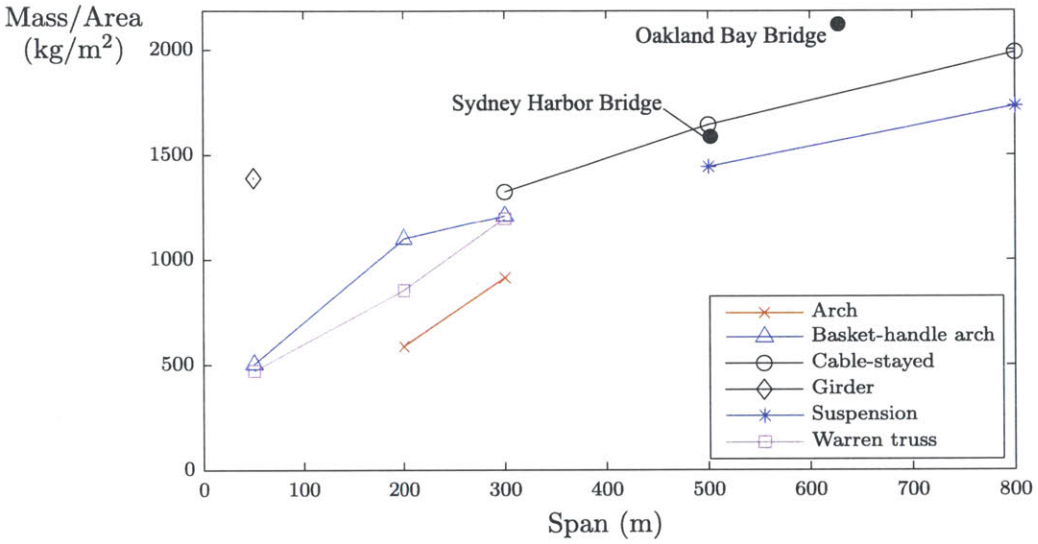


Figure 4.1 – Plotting mass per unit deck area against span shows an approximately linear trend, with values comparable to real bridges.

These data modifications are used only for plotting Fig. 4.1, and are discarded for the remainder of the dissertation.

The material per unit area generally increases with span—an intuitively sensible trend from an engineering perspective. The cable-stayed bridges require relatively more material than other systems. An examination of the data show that much of their mass is used to stiffen the towers against bending under asymmetric live load on the main span only. Our simplified representation of the towers is likely responsible for this.

The values of mass per unit area are comparable to existing structures. The superstructure and deck elements of the Sydney Harbour arch bridge in Australia, for example, weigh 39×10^6 kg (neglecting the weight of the bridge's backspans and towers). Considering its main span of 503 m, and its deck width of 49 m, this gives the Sydney bridge a mass per unit area of 1580 kg/m^2 [Pylon Lookout, 2003]. This places the bridge between the cable-stayed and suspension bridge curves, and approximately along a continuation of the arch bridge curve, in Fig. 4.1.

The self-anchored suspension bridge constructed in 2013 to replace part of the Oakland-to-San Francisco Bay Bridge uses 67×10^6 kg of steel in its superstructure and has a main

span of length 624 m and width 50 m, leading to a mass per unit deck area of 2147 kg/m^2 [California Department of Transport, 2012]. This is approximately in the region of the cable-stayed and suspension curves in Fig. 4.1. The extra superstructure weight is to be expected in a self-anchored bridge; lighter foundations would offset this weight gain in a broader-ranging analysis.

4.2 Examination of representative designs

Analyzing overall trends is a useful way to understand results at a high level, but is limited by its lack of granularity. The previous section's material usage plot considers only the lightest design for each bridge model, offering no insight into the range of solutions reached by various algorithms. This section explores those ranges.

Engineering optimization papers usually examine their results from the perspective of the application domain; validation of results in this way tests their rationality and the rationality of the problem formulation, as well as demonstrating the practical applicability of the optimization method. The large number of distinct designs in this work, however, makes individual examination of each one impractical, and necessitates the reduction of the data to a manageable subset. We accordingly use *k-means clustering* to partition the data into a few groups of similar designs, from which representative designs are selected and examined.

The goals of this exercise are to understand of the range of designs in the data and, through this, to build a general picture of the character of the design spaces and the differences among algorithms' solutions to identical or very similar problems.

4.2.1 Clustering methodology

Clustering groups bridge designs such that those in each cluster are similar, while the clusters themselves are widely spaced. Halkidi et al. [2001] give an overview of the clustering concepts and terms used here.

The first requirements for any clustering methodology are a set of design-identifying features and a measure of the distance between designs. Having chosen and run a clustering algorithm, its output can then be evaluated using one of the many proposed validation techniques.

Since this exercise seeks to group physically similar structures, the design variables (§3.1.2) are used as identifying features, and a separate clustering is carried out for each of the study's fourteen models. So that design variables contribute approximately equally to the

identification of a design, we normalize across designs for a given structure such that each variable has zero mean and unit standard deviation.

The clustering outcome is unpredictably sensitive to the measurement of distance between designs. All clustering operations are therefore repeated with four different measures of inter-design distance—*Squared Euclidean*, *City Block*, *Cosine*, and *Correlation*—and the best from among the four resulting groupings is used.

As is well-established in the literature, the choice of clustering algorithm has a major effect on the outcome, and different algorithms are better suited to different tasks. We choose the partitional k-means algorithm for its computational efficiency and robustness [MacQueen, 1967]. The algorithm is essentially an iterative minimization of the quantity E (Eq. 4.1).

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} d(\vec{x}, m_i) \tag{4.1}$$

where k is the pre-specified number of clusters, m_i is the center of cluster C_i, and $d(\vec{x}, m_i)$ is the distance between the design $\vec{x}$ and m_i. The algorithm begins with a set of k randomly-generated cluster centers, assigns each point to its closest center, and then recomputes the centers. It terminates when the centers stop changing. The algorithm runs twenty times, each with a different randomly-generated initial set of cluster centers, and chooses the results that minimize E.

The remaining input parameters are the number of clusters, k, and the distance measure, d. The clustering is repeated using all four previously-mentioned distance measures, and for all values of k between 2 and 6, resulting in twenty possible groupings of designs. The best grouping is determined using one of many available clustering validation methods, which can either be *external* or *internal*.

External clustering validation requires a set of labels, typically generated by expert users, which identify each design as a member of a particular class. The assignment of these labels would, in this case, be prohibitively time-consuming and subjective; we look instead to internal validation metrics.

Under internal validation, a good solution has high intra-cluster density (the members of each cluster are close to each other) and low inter-cluster density (the clusters themselves are far apart). A number of metrics can evaluate these characteristics; this work uses the

Silhouette index $S(k, d)$ (Eq. 4.2) [Liu et al., 2010].

$$S(k,d) = \frac{1}{k}\left[\sum_{i=1}^{k}\left(\frac{1}{n_i}\sum_{x\in C_i}\frac{b_d(\vec{x}) - a_d(\vec{x})}{\max(b_d(\vec{x}), a_d(\vec{x})}\right)\right] \tag{4.2}$$

$$[k^*, d^*] = \text{argmax } S(k,d)$$

where k is the number of clusters; d represents the method used to measure distance between designs; n_i is the number of designs in the i^{th} cluster C_i; $a_d(\vec{x})$ is the mean distance from each design $\vec{x}$ to the other points in the same cluster, and $b_d(\vec{x})$ is the minimum mean distance from $\vec{x}$ to points in a different cluster, minimized over clusters.

The k-means algorithm is run for all combinations of $k = 2 \ldots 6$ and $d \in$ {Squared Euclidean, City Block, Cosine, Correlation} to determine the S-maximizing k^* and d^*, and their associated grouping of the designs for a given bridge model.

4.2.2 Representative designs

Having found the best grouping of designs using the iterative k-means scheme, a single design is selected at random from each cluster using a uniform probability distribution, forming a representative subset.

Table 4.1 summarizes the clustering output for each of the fourteen structures, specifying the number of clusters k^*, the distance metric d^*, and the silhouette index S. The table's rightmost column references the figures that display the selected designs, and the following sections describe each design in turn.

These designs, it should be noted, are not all solutions to the same problem. The clustering scheme groups *all* designs for each structural type, regardless of whether they emerged from tightly- or loosely-constrained problems, or from mass or stiffness optimizations. A lighter representative solution is not, therefore, necessarily better than a heavier one, since the latter may be an answer to a more onerous problem.

A note on visual and numerical representation

The design variable values for each representative design are shown alongside an isometric projection of the model generated using the GSA software. The physical parameters controlled by these variables are noted in a table at the start of each section. Chapter 3 provides sufficient details to reproduce these GSA models from the design variable values.

Table 4.1 – Summary of clustering output, showing the final number of clusters k^*, the distance metric d^*, and the silhouette value S. The k^* selected designs for each structure are shown in the referenced figures.

Structure Type	Span	k^*	d^*	$S(k^*, d^*)$	See Fig. :
Trussed arch	200 m	2	Sq. Euclidean	0.84	4.2
	300 m	6	Cosine	0.68	4.3
Basket-handle arch	50 m	5	Correlation	0.75	4.4
	200 m	6	Correlation	0.56	4.5
	300 m	2	Cosine	0.61	4.6
Cable-stayed	300 m	3	Correlation	0.55	4.7
	500 m	2	Sq. Euclidean	0.72	4.8
	800 m	2	Correlation	0.66	4.9
Girder	500 m	4	Cosine	0.98	4.10
Suspension	500 m	6	Correlation	0.55	4.11
	800 m	5	Cosine	0.56	4.12
Warren truss	50 m	2	Cosine	0.66	4.13
	200 m	2	Correlation	0.75	4.14
	300 m	4	Cosine	0.55	4.15

Each design is accompanied by a set of symbols denoting key attributes of the optimization problem they solve. These sets contain three elements. The first denotes the optimization problem's objective ("M" for mass or "S" for stiffness); the second indicates the types of constraints present ("D_S" and "D_A" for displacement constraints under symmetric and asymmetric loading, "S" for stress constraints on all elements, and "M" for mass constraints); and the final symbol indicates whether the design variables control geometry and sizing ("G") or sizing only ("S"). For example, the set {M | D_S S | G} indicates a mass minimization problem with displacement constraints under symmetric loading, stress constraints on all elements, and variable geometry. This information provides useful context for interpreting the results.

Trussed arches

Representative designs from the two classes of trussed arch are examined—one spanning 200 m, shown in Fig. 4.2, and the other spanning 300 m, shown in Fig. 4.3. Table 4.2 specifies the physical parameters which the design variables control in accordance with Chapter 3.

The lighter of the two 200 m trussed arches (Fig. 4.2a) has a much higher inner arch

A_1	$0.05\,\mathrm{m}^2$
A_2	$0.05\,\mathrm{m}^2$
A_3	$0.15\,\mathrm{m}^2$
A_4	$0.18\,\mathrm{m}^2$
A_5	$0.05\,\mathrm{m}^2$
A_6	$0.10\,\mathrm{m}^2$
A_7	$0.01\,\mathrm{m}^2$
A_8	$0.10\,\mathrm{m}^2$
A_9	$0.01\,\mathrm{m}^2$
A_{10}	$0.01\,\mathrm{m}^2$
A_{11}	$0.05\,\mathrm{m}^2$
A_{12}	$0.01\,\mathrm{m}^2$
h_{in}	50.0 m
d	10.0 m
Mass	4.67×10^6 kg
S | M S | G	

(a)

A_1	$0.15\,\mathrm{m}^2$
A_2	$0.15\,\mathrm{m}^2$
A_3	$0.49\,\mathrm{m}^2$
A_4	$0.38\,\mathrm{m}^2$
A_5	$0.18\,\mathrm{m}^2$
A_6	$0.28\,\mathrm{m}^2$
A_7	$0.03\,\mathrm{m}^2$
A_8	$0.11\,\mathrm{m}^2$
A_9	$0.01\,\mathrm{m}^2$
A_{10}	$0.01\,\mathrm{m}^2$
A_{11}	$0.10\,\mathrm{m}^2$
A_{12}	$0.01\,\mathrm{m}^2$
h_{in}	35.0 m
d	5.0 m
Mass	9.67×10^6 kg
S | M S | S	

(b)

Figure 4.2 – Design **(a)**'s higher arch rise, h_{in}, and greater separation between the inner and outer arches,d, allows it to have smaller arch rib and deck sections (A_4 and A_3).

rise, h_{in}, and a greater separation between the inner and outer arches, d. Thanks to the stiffness it derives from this geometry, its structural elements, especially the outer arch ribs and the transverse deck beams (A_3 and A_4), are considerably smaller in cross-section. The heavier design is a solution to a sizing optimization problem, meaning its geometric design variables (h_{in} and d) are fixed at 35.0 m and 5.0 m.

The six representative 300 m trussed arches (Fig. 4.3) can be loosely classified by overall geometry into three deep arches (on the left of the figure) and three shallow arches (on

Table 4.2 – Trussed arch design variables, used to specify the representative designs in Figs. 4.2 and 4.3.

A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	
A_4	Inner arch ribs	Superstructure
A_5	Outer arch ribs	
A_6	Truss verticals	
A_7	Truss diagonals below deck	
A_8	Truss diagonals above deck	
A_9	Truss bottom chord	
A_{10}	Lateral transverse braces	
A_{11}	Deck tension hangers	
A_{12}	Lateral diagonal braces	
h_{in}	Rise of inner arch	
d	Midspan vertical spacing between arches	

A_1	$0.07\,\mathrm{m}^2$
A_2	$0.02\,\mathrm{m}^2$
A_3	$0.27\,\mathrm{m}^2$
A_4	$0.65\,\mathrm{m}^2$
A_5	$0.09\,\mathrm{m}^2$
A_6	$0.01\,\mathrm{m}^2$
A_7	$0.03\,\mathrm{m}^2$
A_8	$0.09\,\mathrm{m}^2$
A_9	$0.08\,\mathrm{m}^2$
A_{10}	$0.01\,\mathrm{m}^2$
A_{11}	$0.03\,\mathrm{m}^2$
A_{12}	$0.01\,\mathrm{m}^2$
h_{in}	99.3 m
d	9.1 m
Mass	1.00×10^7 kg
S | D_A	D_S S M | G

(a)

A_1	$0.03\,\mathrm{m}^2$
A_2	$0.04\,\mathrm{m}^2$
A_3	$0.02\,\mathrm{m}^2$
A_4	$0.19\,\mathrm{m}^2$
A_5	$0.04\,\mathrm{m}^2$
A_6	$0.01\,\mathrm{m}^2$
A_7	$0.04\,\mathrm{m}^2$
A_8	$0.01\,\mathrm{m}^2$
A_9	$0.09\,\mathrm{m}^2$
A_{10}	$0.01\,\mathrm{m}^2$
A_{11}	$0.04\,\mathrm{m}^2$
A_{12}	$0.01\,\mathrm{m}^2$
h_{in}	52.5 m
d	10.0 m
Mass	3.52×10^6 kg
M | D_A	D_S S | S

(b)

A_1	$0.05\,\mathrm{m}^2$
A_2	$0.05\,\mathrm{m}^2$
A_3	$0.80\,\mathrm{m}^2$
A_4	$0.25\,\mathrm{m}^2$
A_5	$0.05\,\mathrm{m}^2$
A_6	$0.02\,\mathrm{m}^2$
A_7	$0.10\,\mathrm{m}^2$
A_8	$0.45\,\mathrm{m}^2$
A_9	$0.10\,\mathrm{m}^2$
A_{10}	$0.01\,\mathrm{m}^2$
A_{11}	$0.10\,\mathrm{m}^2$
A_{12}	$0.01\,\mathrm{m}^2$
h_{in}	120.0 m
d	10.0 m
Mass	1.97×10^7 kg
M | D_A	D_S S | S

(c)

A_1	$0.00\,\mathrm{m}^2$
A_2	$2.37\,\mathrm{m}^2$
A_3	$0.00\,\mathrm{m}^2$
A_4	$0.00\,\mathrm{m}^2$
A_5	$0.00\,\mathrm{m}^2$
A_6	$0.00\,\mathrm{m}^2$
A_7	$0.07\,\mathrm{m}^2$
A_8	$0.00\,\mathrm{m}^2$
A_9	$0.00\,\mathrm{m}^2$
A_{10}	$0.00\,\mathrm{m}^2$
A_{11}	$0.00\,\mathrm{m}^2$
A_{12}	$0.00\,\mathrm{m}^2$
h_{in}	52.5 m
d	10.0 m
Mass	1.00×10^7 kg
S | M | S	

(d)

A_1	$0.02\,\mathrm{m}^2$
A_2	$0.03\,\mathrm{m}^2$
A_3	$0.60\,\mathrm{m}^2$
A_4	$0.10\,\mathrm{m}^2$
A_5	$0.02\,\mathrm{m}^2$
A_6	$0.42\,\mathrm{m}^2$
A_7	$0.01\,\mathrm{m}^2$
A_8	$0.34\,\mathrm{m}^2$
A_9	$0.02\,\mathrm{m}^2$
A_{10}	$0.00\,\mathrm{m}^2$
A_{11}	$0.01\,\mathrm{m}^2$
A_{12}	$0.00\,\mathrm{m}^2$
h_{in}	78.2 m
d	5.7 m
Mass	1.69×10^7 kg
M | D_A	D_S S | G

(e)

A_1	$0.01\,\mathrm{m}^2$
A_2	$0.00\,\mathrm{m}^2$
A_3	$0.66\,\mathrm{m}^2$
A_4	$0.15\,\mathrm{m}^2$
A_5	$0.02\,\mathrm{m}^2$
A_6	$0.43\,\mathrm{m}^2$
A_7	$0.01\,\mathrm{m}^2$
A_8	$0.21\,\mathrm{m}^2$
A_9	$0.03\,\mathrm{m}^2$
A_{10}	$0.00\,\mathrm{m}^2$
A_{11}	$0.01\,\mathrm{m}^2$
A_{12}	$0.00\,\mathrm{m}^2$
h_{in}	52.5 m
d	10.0 m
Mass	1.47×10^7 kg
S | M S | S	

(f)

Figure 4.3 – Each of the six representative 300 m trussed arches is, in its own way a rational design. The shallower arches on the right of the figure are, on average, lighter than the deeper ones, and the material distribution is different in each case.

the right of the figure). The lightest design (Fig. 4.3b) balances its material distribution between the arch superstructure and the deck, while the other two heavier, shallow arches (Figs. 4.3d and 4.3e) concentrate material in the deck. The lightest of the deep arches (Fig. 4.3a) has a large inner arch cross-section, while the other two place more material in the truss diagonals and verticals (A_6 and A_8).

Each of these has a different design intent and is, in its own way, a rational outcome. Different designs emerge as solutions to relatively similar problems, likely indicating a design space with various local optima and optimization algorithms that are inclined to converge on those local optima.

Basket-handle arches

The selected designs for the three basket-handle arch types (50 m, 200 m, and 300 m) are shown in Figs. 4.4, 4.5, and 4.6; Table 4.3 identifies their design variables.

Three of the five selected 50 m arch bridges have the same span-to-rise ratio of 5.0, but different material distributions (Figs. 4.4b, 4.4d, and 4.4d). The lightest of these three (Fig. 4.4b) balances its material between arch ribs and deck elements, while the heavier two tilt the material distribution strongly towards either the arch or the deck.

The other two designs have very different span-to-rise ratios: a shallow $\gamma_z = 20.0$ (Fig. 4.4a) and a deep $\gamma_z = 4.0$ (Fig. 4.4c). The lighter, and deeper, of the two has relatively light structural elements (Fig. 4.4c), whereas the heavier, shallow one has very heavy elements to compensate for the lack of stiffness derived from its geometry. These designs, despite being quite different in character, are locally-optimal solutions to very similar design problems. (The additional stress constraints imposed on the heavier design are all inactive at convergence.) The existence of multiple locally-optimal solutions with

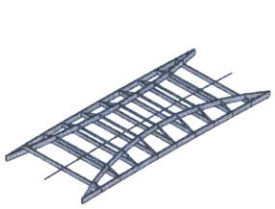
Table 4.3 – The basket-handle arch design variables are used to specify designs in Figs. 4.4, 4.5, and 4.3, following Chapter 3's specifications.

A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	
A_7	Deck diagonal beams	
A_4	Arch ribs	Superstructure
A_5	Deck hangers	
A_6	Lateral braces	
γ_z	Span-to-rise ratio of arch	
γ_y	Span-to-inward lean ratio of arch	

very different masses indicates the complexity of the design space, and the downside of being unable to escape local optima.

The six naturally-emerging clusters of 200 m basket-handle arch designs can be further split into two groups based on geometry: the representative designs in Figs. 4.5a through 4.5c with high arch rises ($\gamma_z = 4.0$), and those in Figs. 4.5d through 4.5f, which are much shallower.

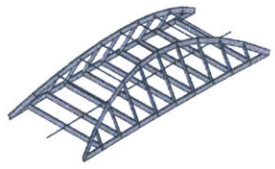
The lightest of the designs (Fig. 4.5a) is an intuitively rational solution to a displacement-constrained mass minimization. In response to the requirement to resist asymmetric live loading, more material is concentrated in the deck elements (A_1, A_2, A_3, and A_7) than in the main arch (A_4, A_5, and A_6). The other two deep arches emerge as solutions to mass-constrained stiffness minimization problems (Figs. 4.5b and 4.5c). Unsurprisingly,



A_1	0.19 m^2
A_2	0.00 m^2
A_3	0.06 m^2
A_4	0.18 m^2
A_5	0.02 m^2
A_6	0.01 m^2
γ_z	20.0
γ_y	10.0
Mass	3.91 × 10^5 kg

M | D_A D_S S | S

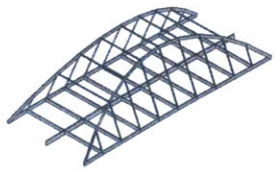
(a)



A_1	0.14 m^2
A_2	0.00 m^2
A_3	0.16 m^2
A_4	0.15 m^2
A_5	0.02 m^2
A_6	0.00 m^2
γ_z	5.0
γ_y	10.0
Mass	5.00 × 10^5 kg

S | D_A D_S M | S

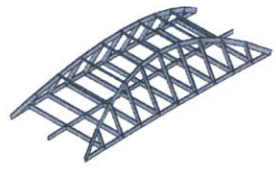
(b)



A_1	0.03 m^2
A_2	0.07 m^2
A_3	0.03 m^2
A_4	0.02 m^2
A_5	0.00 m^2
A_6	0.00 m^2
γ_z	4.0
γ_y	10.0
Mass	1.21 × 10^5 kg

M | D_A D_S | S

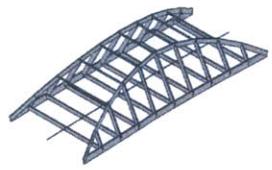
(c)



A_1	0.06 m^2
A_2	0.05 m^2
A_3	0.10 m^2
A_4	0.10 m^2
A_5	0.05 m^2
A_6	0.01 m^2
γ_z	5.0
γ_y	10.0
Mass	3.61 × 10^5 kg

S | M S | S

(d)



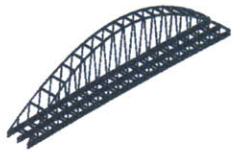
A_1	0.35 m^2
A_2	0.00 m^2
A_3	0.06 m^2
A_4	0.08 m^2
A_5	0.06 m^2
A_6	0.01 m^2
γ_z	5.0
γ_y	10.0
Mass	4.99 × 10^5 kg

S | M | S

(e)

Figure 4.4 – The deepest of the selected basket-handle arches, **(c)**, is also the lightest. Three designs that have the same γ_z—**(b)**, **(c)**, and **(e)**—have very different material distributions.

they are heavier than the light design in Fig. 4.5b. The latter of these two, despite the presence of additional displacement constraints, is heavier than the former, needlessly concentrating material in the arch bracing and hangers (A_5 and A_6).



A_1	$0.21\,\mathrm{m}^2$
A_2	$0.11\,\mathrm{m}^2$
A_3	$0.06\,\mathrm{m}^2$
A_4	$0.11\,\mathrm{m}^2$
A_5	$0.01\,\mathrm{m}^2$
A_6	$0.14\,\mathrm{m}^2$
A_7	$0.07\,\mathrm{m}^2$
γ_z	4.0
γ_y	40.0
Mass	3.10×10^6 kg

M | D_A D_S | S

(a)

A_1	$1.10\,\mathrm{m}^2$
A_2	$1.82\,\mathrm{m}^2$
A_3	$0.70\,\mathrm{m}^2$
A_4	$4.65\,\mathrm{m}^2$
A_5	$0.63\,\mathrm{m}^2$
A_6	$1.77\,\mathrm{m}^2$
A_7	$2.09\,\mathrm{m}^2$
γ_z	4.0
γ_y	40.0
Mass	5.29×10^7 kg

S | M S | S

(b)

A_1	$1.22\,\mathrm{m}^2$
A_2	$0.26\,\mathrm{m}^2$
A_3	$0.89\,\mathrm{m}^2$
A_4	$3.13\,\mathrm{m}^2$
A_5	$0.31\,\mathrm{m}^2$
A_6	$0.08\,\mathrm{m}^2$
A_7	$0.34\,\mathrm{m}^2$
γ_z	4.0
γ_y	40.0
Mass	3.00×10^7 kg

S | D_A D_S M S | S

(c)

A_1	$2.00\,\mathrm{m}^2$
A_2	$0.11\,\mathrm{m}^2$
A_3	$0.58\,\mathrm{m}^2$
A_4	$3.65\,\mathrm{m}^2$
A_5	$0.14\,\mathrm{m}^2$
A_6	$0.01\,\mathrm{m}^2$
A_7	$0.26\,\mathrm{m}^2$
γ_z	20.0
γ_y	13.3
Mass	2.99×10^7 kg

S | D_A D_S M | S

(d)

A_1	$0.24\,\mathrm{m}^2$
A_2	$0.18\,\mathrm{m}^2$
A_3	$0.05\,\mathrm{m}^2$
A_4	$0.31\,\mathrm{m}^2$
A_5	$0.02\,\mathrm{m}^2$
A_6	$0.03\,\mathrm{m}^2$
A_7	$0.57\,\mathrm{m}^2$
γ_z	12.5
γ_y	13.3
Mass	6.30×10^6 kg

S | D_A D_S M S | S

(e)

A_1	$2.33\,\mathrm{m}^2$
A_2	$2.16\,\mathrm{m}^2$
A_3	$0.10\,\mathrm{m}^2$
A_4	$1.00\,\mathrm{m}^2$
A_5	$0.10\,\mathrm{m}^2$
A_6	$0.05\,\mathrm{m}^2$
A_7	$0.80\,\mathrm{m}^2$
γ_z	12.5
γ_y	13.3
Mass	2.97×10^7 kg

S | M | S

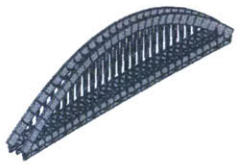
(f)

Figure 4.5 – The first three representative 200 m basket-handle arches have the same geometry, but very different material distributions. The last three structures have shallow arches and heavy elements.

The 300 m arch designs form two clusters (Fig. 4.6). The first is a light, deep bridge, with most of the material concentrated in the deck to resist asymmetric lateral loads. The design representing the second cluster solves a stiffness maximization problem and is, accordingly, much heavier.

A_1	$0.28\,\mathrm{m}^2$
A_2	$0.00\,\mathrm{m}^2$
A_3	$0.05\,\mathrm{m}^2$
A_4	$0.15\,\mathrm{m}^2$
A_5	$0.01\,\mathrm{m}^2$
A_6	$0.02\,\mathrm{m}^2$
A_7	$0.09\,\mathrm{m}^2$
γ_z	3.4
γ_y	51.0
Mass	4.74×10^6 kg
M |	D_A D_S | G

(a)



A_1	$4.81\,\mathrm{m}^2$
A_2	$1.59\,\mathrm{m}^2$
A_3	$2.58\,\mathrm{m}^2$
A_4	$6.96\,\mathrm{m}^2$
A_5	$0.68\,\mathrm{m}^2$
A_6	$0.14\,\mathrm{m}^2$
A_7	$2.88\,\mathrm{m}^2$
γ_z	4.0
γ_y	80.0
Mass	1.45×10^8 kg
S |	M S | S

(b)

Figure 4.6 – The 300 m basket-handle arches are generally quite deep. Some use light elements ([**a**]), but others are much heavier ([**b**]).

Cable-stayed bridges

Selected designs for the three cable-stayed bridge types are shown in Figs. 4.7, 4.8, and 4.9. The physical parameters controlled by the design variables are noted in Table 4.4.

The three selected designs for the 300 m cable-stayed bridges are most naturally categorized by height. The tallest of them (Fig. 4.7a) is the lightest, and also has the smallest cable area. This makes good engineering sense; the increased truss height means a cable of a given cross-section is vertically stiffer, requiring less cable material. The next tallest bridge (Fig. 4.7b) has much thicker cables supporting the deck. The least tall bridge (Fig. 4.7c), interestingly, has quite small cables, but a much stiffer deck. This, again, is physically rational. The vertical stiffness gained by adding material to such near-horizontal cables makes this an ineffective system when the towers are so short, and it is better to instead stiffen the deck.

The two selected 500 m bridges have almost the exact same weight, though they achieve it in different ways. The taller bridge (Fig. 4.8a) has slender vertically-oriented cables and light deck elements. The forces in these cables, however, require large tower cross-sections to resist bending in the towers under asymmetric loads. The shorter bridge has substantially bigger deck elements (especially the longitudinal deck beams, A_1 and A_2) to compensate for its cables lack of vertical stiffness.

The grouping of the 800 m cable-stayed bridges presents a similar picture: a taller structure with lighter elements and a shorter structure with heavier elements—in this case both the deck and cable elements—to compensate.

Table 4.4 – Cable-stayed bridge design variables, used to specify designs in Figs. 4.7, 4.8, and 4.9.

A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	
A_6	Deck truss diagonals	
A_4	Cables	Superstructure
A_5	Towers posts	
h	Tower height	

A_1	$0.05\,\mathrm{m}^2$
A_2	$0.03\,\mathrm{m}^2$
A_3	$0.05\,\mathrm{m}^2$
A_4	$0.02\,\mathrm{m}^2$
A_5	$5.00\,\mathrm{m}^2$
A_6	$0.03\,\mathrm{m}^2$
h	90.0 m
Mass	2.33×10^7 kg

M | D_A D_S | S

(a)

A_1	$0.06\,\mathrm{m}^2$
A_2	$0.03\,\mathrm{m}^2$
A_3	$0.06\,\mathrm{m}^2$
A_4	$0.57\,\mathrm{m}^2$
A_5	$7.00\,\mathrm{m}^2$
A_6	$0.03\,\mathrm{m}^2$
h	60.0 m
Mass	5.00×10^7 kg

S | M | S

(b)

A_1	$0.18\,\mathrm{m}^2$
A_2	$0.09\,\mathrm{m}^2$
A_3	$0.03\,\mathrm{m}^2$
A_4	$0.07\,\mathrm{m}^2$
A_5	$7.73\,\mathrm{m}^2$
A_6	$0.05\,\mathrm{m}^2$
h	30.0 m
Mass	2.41×10^7 kg

M | D_A D_S S | S

(c)

Figure 4.7 – The three representative 300 m cable-stayed bridges are very different. The tallest, and lightest, of them—**(a)**—has thin, efficient cables, while the less tall bridges compensate with stiffer decks or thicker cables.

A_1	$0.12\,\mathrm{m}^2$
A_2	$0.09\,\mathrm{m}^2$
A_3	$0.11\,\mathrm{m}^2$
A_4	$0.05\,\mathrm{m}^2$
A_5	$9.52\,\mathrm{m}^2$
A_6	$0.12\,\mathrm{m}^2$
h	200.0 m
Mass	1.00×10^8 kg

S | D_A D_S M | S

(a)

A_1	$1.28\,\mathrm{m}^2$
A_2	$1.10\,\mathrm{m}^2$
A_3	$0.06\,\mathrm{m}^2$
A_4	$0.03\,\mathrm{m}^2$
A_5	$5.00\,\mathrm{m}^2$
A_6	$0.40\,\mathrm{m}^2$
h	100.0 m
Mass	1.00×10^8 kg

S | M | S

(b)

Figure 4.8 – The two 500 m cable-stayed designs achieve the same mass in different ways. **(a)** has tall towers, vertically-oriented slender cables, and light deck elements, while **(b)** has shorter towers and heavy deck elements to compensate.

A_1	$0.10\,\mathrm{m}^2$
A_2	$0.10\,\mathrm{m}^2$
A_3	$0.10\,\mathrm{m}^2$
A_4	$0.03\,\mathrm{m}^2$
A_5	$5.00\,\mathrm{m}^2$
A_6	$0.10\,\mathrm{m}^2$
h	240.0 m
Mass	7.64×10^7 kg

M | D_A D_S | S

A_1	$1.71\,\mathrm{m}^2$
A_2	$0.10\,\mathrm{m}^2$
A_3	$0.10\,\mathrm{m}^2$
A_4	$0.32\,\mathrm{m}^2$
A_5	$5.00\,\mathrm{m}^2$
A_6	$0.23\,\mathrm{m}^2$
h	200.0 m
Mass	2.61×10^8 kg

S | D_A D_S M S | S

(a)

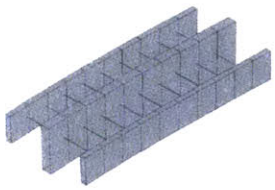
Figure 4.9 – The 800 m cable-stayed designs, like the 500 m ones, group into tall bridges with light elements ([**a**]) and shorter bridges with heavier elements ([**b**]).

Steel girder bridges

The four selected designs for the 50 m steel girder bridge, whose design variables are named in Table 4.5, are shown in Fig. 4.10. The lightest of them (Fig. 4.10b) is likely a solution to a highly-unconstrained problem. The other three have similar masses, but achieve it by stiffening either one of the longitudinal beams more than the other (Figs.4.10a and 4.10d) or by a more balanced approach (Fig. 4.10c).

Table 4.5 – Girder bridge design variables, used to specify designs in Fig. 4.10.

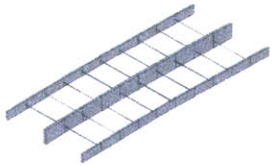
A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	



A_1	$3.18\,\mathrm{m}^2$
A_2	$6.37\,\mathrm{m}^2$
A_3	$0.00\,\mathrm{m}^2$
Mass	5.00×10^6 kg

S | M | S

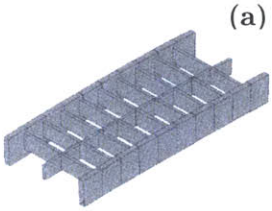
(a)



A_1	$0.67\,\mathrm{m}^2$
A_2	$1.28\,\mathrm{m}^2$
A_3	$0.00\,\mathrm{m}^2$
Mass	1.03×10^6 kg

M | D_A D_S | S

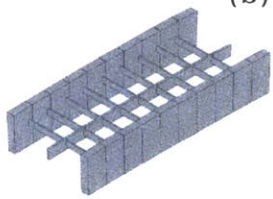
(b)



A_1	$3.12\,\mathrm{m}^2$
A_2	$1.36\,\mathrm{m}^2$
A_3	$1.42\,\mathrm{m}^2$
Mass	4.98×10^6 kg

S | D_A D_S M | S

(c)



A_1	$5.14\,\mathrm{m}^2$
A_2	$0.94\,\mathrm{m}^2$
A_3	$0.42\,\mathrm{m}^2$
Mass	5.00×10^6 kg

S | M | S

(d)

Figure 4.10 – Selected 50 m steel girder bridge designs.

Suspension bridges

The six selected 500 m suspension bridges show dramatic geometric and sizing variation. The lightest design is also the shallowest (Fig. 4.11a). The two deepest designs (Figs. 4.11a and 4.11b) distribute material towards the diagonal deck elements (A_6) and the primary cables (A_4), respectively.

Four of the 800 m suspension bridges (Fig. 4.12) have remarkably similar mass, to within a few hundredths of a percent, although their designs are quite distinct. This indicates a design space with many similarly-good local optima. Fig. 4.12c's design, for example, has the tallest towers and therefore gets the most stiffness from each unit of material in its primary cables. Fig. 4.12e's, on the other hand, has shorter towers and derives its stiffness from deck elements, especially the deck diagonals (A_6).

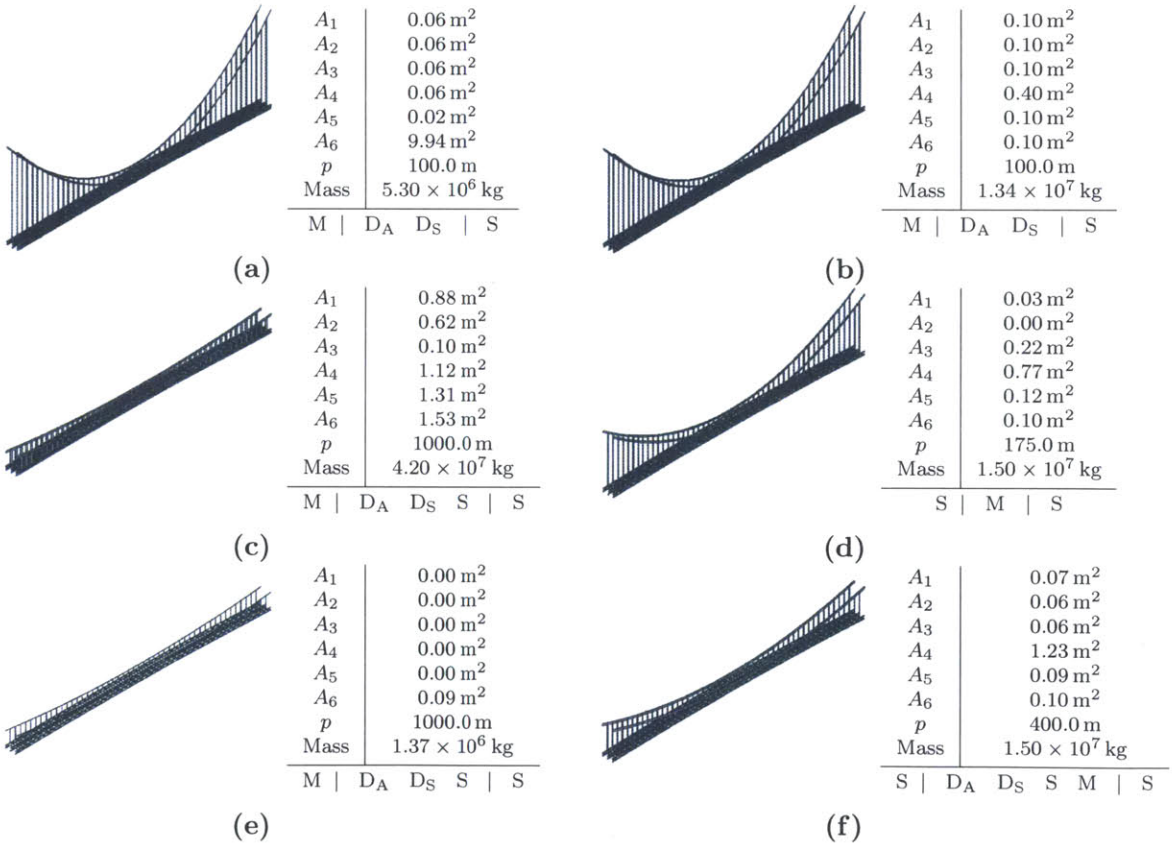


Figure 4.11 – The six selected 500 m suspension bridge designs vary widely in their material distribution and geometry.

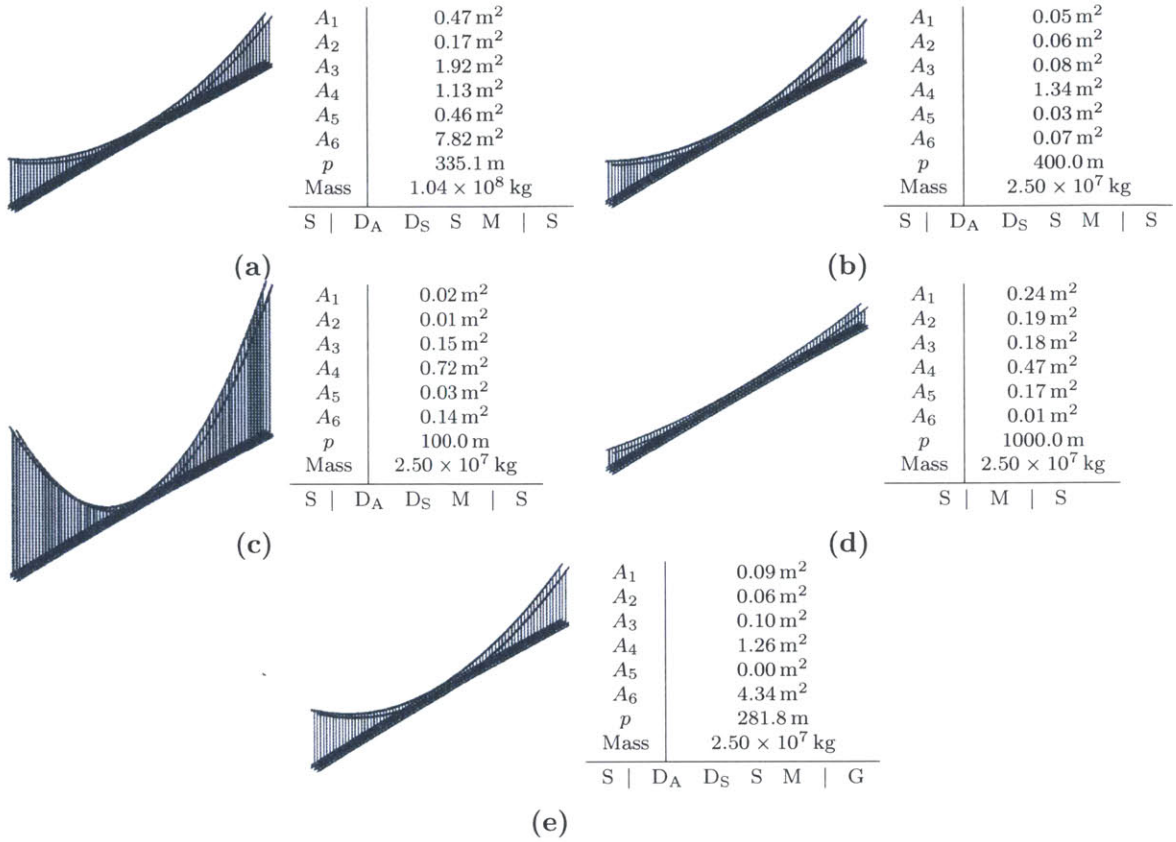


Figure 4.12 – Four of the selected 800 m suspension bridge designs have remarkably similar mass, despite having distinct design characteristics.

Table 4.6 – Suspension bridge design variables, used to specify designs in Figs. 4.11 and 4.12.

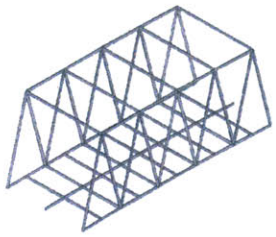
A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	
A_6	Deck truss diagonals	
A_4	Main cables	Superstructure
A_5	Deck hangers	
p	Cable parabola focus (Eq. 3.7)	

Warren truss bridges

Two of the Warren truss categories—50 m and 200 m—are grouped into deep structures with slender elements and shorter structures with thick elements (Figs. 4.13 and 4.14). Both are rational design approaches, but the deeper structures are dramatically lighter in both cases. The 300 m designs, like the 800 m suspension bridges, have very similar mass despite having quite different designs. The deeper trusses have, overall, much more slender elements.

Table 4.7 – Warren truss design variables, used to specify designs in Figs. 4.13, 4.14, and 4.15.

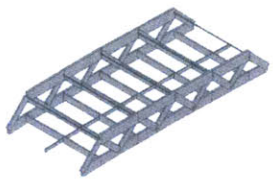
A_1	Outer longitudinal beams	Deck
A_2	Inner longitudinal beams	
A_3	Transverse beams	
A_4	Truss top chord	Superstructure
A_5	Truss diagonals	
A_6	Lateral braces	
h	Inter-chord height	



A_1	$0.02\,\mathrm{m}^2$
A_2	$0.02\,\mathrm{m}^2$
A_3	$0.02\,\mathrm{m}^2$
A_4	$0.02\,\mathrm{m}^2$
A_5	$0.02\,\mathrm{m}^2$
A_6	$0.02\,\mathrm{m}^2$
h	20.0 m
Mass	1.45×10^5 kg

M | D_A D_S | S

(a)



A_1	$0.72\,\mathrm{m}^2$
A_2	$0.03\,\mathrm{m}^2$
A_3	$0.38\,\mathrm{m}^2$
A_4	$0.84\,\mathrm{m}^2$
A_5	$0.32\,\mathrm{m}^2$
A_6	$0.00\,\mathrm{m}^2$
h	5.0 m
Mass	2.00×10^6 kg

S | M | S

(b)

Figure 4.13 – The 50 m Warren truss designs are grouped into **(a)** deep, light structures with slender elements and **(b)** shallow, heavy structures with thicker elements.

A_1	$10.00\,\mathrm{m}^2$
A_2	$10.00\,\mathrm{m}^2$
A_3	$9.99\,\mathrm{m}^2$
A_4	$10.00\,\mathrm{m}^2$
A_5	$0.01\,\mathrm{m}^2$
A_6	$0.01\,\mathrm{m}^2$
h	5.0 m
Mass	1.07×10^8 kg

M | D_A D_S | S

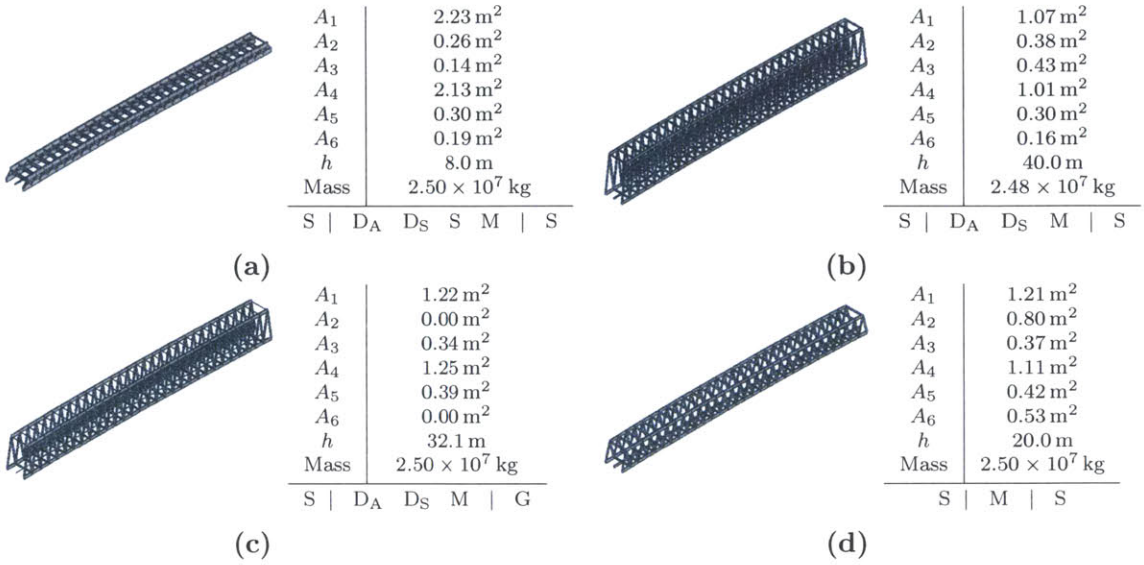
(a)

A_1	$0.50\,\mathrm{m}^2$
A_2	$0.50\,\mathrm{m}^2$
A_3	$0.20\,\mathrm{m}^2$
A_4	$0.40\,\mathrm{m}^2$
A_5	$0.20\,\mathrm{m}^2$
A_6	$0.20\,\mathrm{m}^2$
h	40.0 m
Mass	9.92×10^6 kg

S | S M | S

(b)

Figure 4.14 – The 200 m Warren truss designs are grouped, similarly to the 50 m ones in Fig. 4.13, into deep, light structures and shallow, heavy ones.



(a)

A_1	$2.23\,\mathrm{m}^2$
A_2	$0.26\,\mathrm{m}^2$
A_3	$0.14\,\mathrm{m}^2$
A_4	$2.13\,\mathrm{m}^2$
A_5	$0.30\,\mathrm{m}^2$
A_6	$0.19\,\mathrm{m}^2$
h	8.0 m
Mass	2.50×10^7 kg

S | D_A D_S S M | S

(b)

A_1	$1.07\,\mathrm{m}^2$
A_2	$0.38\,\mathrm{m}^2$
A_3	$0.43\,\mathrm{m}^2$
A_4	$1.01\,\mathrm{m}^2$
A_5	$0.30\,\mathrm{m}^2$
A_6	$0.16\,\mathrm{m}^2$
h	40.0 m
Mass	2.48×10^7 kg

S | D_A D_S M | S

(c)

A_1	$1.22\,\mathrm{m}^2$
A_2	$0.00\,\mathrm{m}^2$
A_3	$0.34\,\mathrm{m}^2$
A_4	$1.25\,\mathrm{m}^2$
A_5	$0.39\,\mathrm{m}^2$
A_6	$0.00\,\mathrm{m}^2$
h	32.1 m
Mass	2.50×10^7 kg

S | D_A D_S M | G

(d)

A_1	$1.21\,\mathrm{m}^2$
A_2	$0.80\,\mathrm{m}^2$
A_3	$0.37\,\mathrm{m}^2$
A_4	$1.11\,\mathrm{m}^2$
A_5	$0.42\,\mathrm{m}^2$
A_6	$0.53\,\mathrm{m}^2$
h	20.0 m
Mass	2.50×10^7 kg

S | M | S

Figure 4.15 – Selected 300 m Warren truss designs

4.3 Discussion

We draw three main conclusions from this chapter's engineering exploration of the results. First, the studied design spaces are complex and variable; they appear to have multiple local optima, which themselves are sensitive to problem formulation. Second, different algorithms converge on different solutions for similar or identical problems. Finally, and most importantly, the data which will be used in the next chapters to study algorithm performance and develop selection techniques are based on valid solutions to real engineering problems.

Within many of the fourteen structure types, different representative designs emerge as solutions to identically- or similarly-posed optimization problems. The non-convexity of simulation-based design spaces in this work, suggested by the observed patterns of representative designs, is not surprising. The nature of this non-convexity, however, is difficult to predict in advance. Some design space's local optima are similarly-valid solutions to the problem; many of the structural types have very different representative designs which use similar amounts of material. Others have representative designs which vary greatly in their material use.

The range of observed solutions also informs our understanding of the algorithms. Using different algorithms to solve the same, or very similar, problems yields vastly different results. The following chapter examines individual algorithm performance closely, but it is safe to assume that some types of algorithm are better suited to certain design spaces

than others and that some algorithms have difficulty escaping local optima.

The broader goal of this dissertation, of course, is to address the algorithm selection problem. In this context, perhaps the most important contribution of this chapter is to verify the resulting designs as valid solutions to realistic structural engineering problems. Given our stated intent of adapting optimization to the realities of practice, this demonstration is particularly important. The next chapter examines trends in algorithm performance, and the following one develops and evaluates automated selection systems. Their results and findings are founded on the underlying engineering validity of the data, enhancing the dissertation's relevance to practical design.

Chapter 5

Results: exploration of algorithm performance

The previous chapter, having verified the optimization-generated designs as rational solutions to real structural engineering problems, ensures that subsequent results and conclusions in the domain of algorithm performance are applicable to practical design scenarios. This chapter examines the data from an algorithm performance perspective, demonstrating solution quality and computational cost trends across the design problems (§5.2) and identifying associations between features of these problems and performance measures for each of the algorithms (§5.3).

§5.2 and §5.3 each answer one of the first two research questions posed at the end of Chapter 1. They improve the general understanding of optimization's true potential and of the importance of the algorithm selection problem, and enable us to broadly recommend situation-specific use of algorithms. Before they do so, §5.1 makes a few notes on the nature of the data and the methods of exploration.

5.1 Preliminary considerations

We use the normalized measures of algorithm performance (§3.2.2) throughout §5.3, to enable inter-problem comparison. Both measures—objective value (quality of the solution) and number of analysis calls (computational efficiency)—lie in the range $[0, 1]$, with values closer to 1 indicating either a better result or a lower computational cost.

5.1.1 Visualization

Where applicable, plots of objective value data are colored orange (lighter when viewed in grayscale) throughout, and computational cost plots are colored blue (darker in grayscale).

A useful tool graphical tool for visualizing ranges of data is the *boxplot*. In this work, boxplots represent a range of numbers with a vertical black line indicating the median, a colored box with outer limits at the range's 25^{th} and 75^{th} percentiles (denoted q_1 and q_3), and two thin horizontal whiskers extend to $4q_1 - 3q_3$ and $4q_3 - 3q_1$. Outlying values beyond the whiskers' range are shown as black dots (Fig. 5.1).

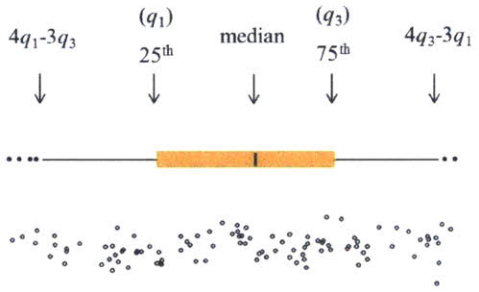


Figure 5.1 – The boxplot is used to visually represent a range of numbers. A vertical black line denotes the median of the range, the box itself stretches from the 25^{th} to the 75^{th} percentiles, and outlying values beyond the black whiskers are shown as black dots.

5.1.2 Patterns of data distribution

Since each problem must have a best-performing and a worst performing algorithm, the distributions of normalized performance have expected peaks at 1 and 0 (Fig. 5.2). When normalized algorithm performance is the dependent variable, as will generally be the case in this chapter and the next, these non-normal distributions preclude the use of many parametric methods for testing association between variables and for predicting outcomes.

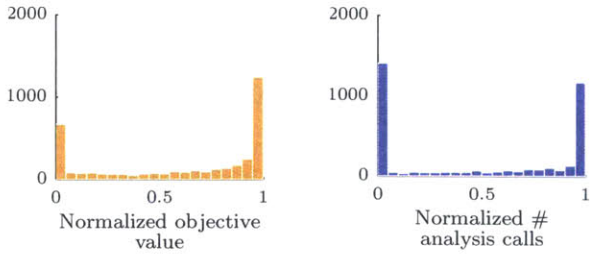


Figure 5.2 – The distribution of both normalized performance measures shows expected concentrations close to values of 0 and 1. These non-normal distributions preclude the use of many parametric tests of association and difference.

In Chapter 6, which uses the data presented here to build predictive systems for algorithm selection, it will sometimes be possible to work with the original, non-normalized

performance data (i.e., the actual values of objective functions and the number of analysis calls). Fig. 5.3 shows their distributions. When problems are split into mass and stiffness optimizations, the natural logarithm of performance measures for all algorithms and problems shows an approximately normal distribution.[1]

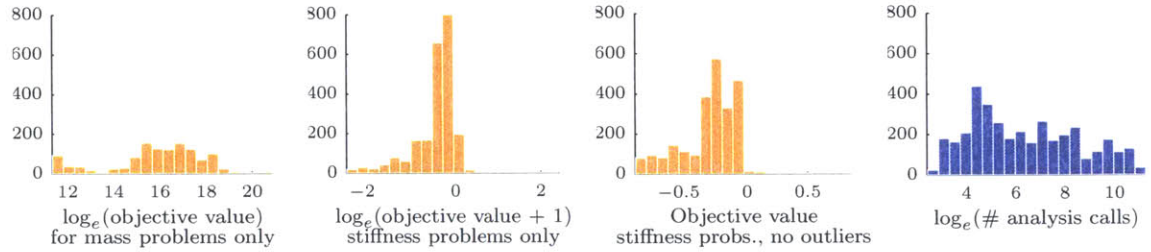


Figure 5.3 – The logarithms of (non-normalized) objective have approximately normal occurrence distributions when split into mass and stiffness problems.[1]

5.1.3 Statistical tests

In §5.3, we seek associations between predictive factors and algorithm performance, or examine differences between mean performance values when problems are grouped in certain ways. Although these associations and differences are shown graphically, it is useful to accompany visualizations with suitable parametric tests. The observed non-normal distributions of performance data promote the use of non-parametric test methods.

To test the hypothesis that groups of data are drawn from samples with significantly different means—useful when evaluating the importance of categorical problem features such as structural type or the nature of design variables—we use the *Kruskal-Wallis One-Way Analysis of Variance by Ranks* test [Kruskal and Wallis, 1952]. The null hypothesis of equal sample means is rejected—and the presence of a statistically significant association is confirmed—when the test's p-value, denoted p_{kw}, is less than 0.05.

When evaluating association between a continuous problem feature and algorithm performance, standard scatterplots are accompanied by *Spearman's Rank Correlation Coefficient*, c_s [Spearman, 1904]. A value of c_s equal to -1 or $+1$ indicates perfectly negative or positive monotonicity, and an associated p-value, p_s, shows that the association between problem features and algorithm performance is statistically different from zero (i.e., that significant correlation exists) if it is less than 0.05.

When drawing conclusions from this chapter's data analysis, we remember that correlation between an algorithm's performance and a problem feature does not mean that this feature

[1] Many stiffness objective values fall in the range $[-1, 0]$; shifting these by 1 enables the calculation of logarithms.

alone is causing the variation. The relationships between algorithm performance and the specified problem features are likely to be nonlinear and dependent on interactions.

5.1.4 Denoting algorithms

Optimization algorithms are referred to by their designated acronyms rather than their full names. These acronyms are repeated for the reader's reference in Table 5.1.

The algorithms fall into two broad categories—local algorithms, which converge when their evaluation of the objective function converges within a tolerance of 1×10^{-4}, possibly on a locally-optimal but globally-suboptimal design; and global algorithms, which search the design space broadly for six hours, return the best feasible design found in that time, and are more likely to find the global optimum.

Table 5.1 – Reminder of the eight algorithms' acronyms

Local	Constrained Optimization by Linear Approximation–*COBYLA* [Powell, 1994]
	Bounded Optimization by Quadratic Approximation–*BOBYQA* [Powell, 2009]
	Nelder-Mead Simplex—*NEL-MEAD* [Nelder and Mead, 1965]
	Subplex—*SUBPLEX* [Rowan, 1990]
	Principal Axis Method—*PR-AXIS* [Brent, 2002]
Global	Dividing Rectangles Method—*DIRECT* [Jones et al., 1993]
	Controlled Random Search—*CRS* [Kaelo and Ali, 2006]
	Improved Stochastic Ranking Evolution Strategy—*ISRES* [Runarsson and Yao, 2005]

5.2 Overall performance of algorithms

The first research question posed in Chapter 1 is: *"How do different optimization algorithms perform on a representative set of realistic structural design problems?"* Chapters 1 and 2 described how the optimization research community's approach to design automation has led to a situation where practical examples are included in the literature as ad-hoc case studies. By neglecting to document how often algorithms produced suboptimal designs or required extensive tweaking before achieving successful results, the field leaves design engineers with an insufficiently-clear picture of the true potential of optimization as a robust generator of high-quality designs.

This section, in answering the first research question, begins to address this limitation of the field by documenting how often each algorithm performs best on a design problem, by examining the overall quality of algorithms' solutions across many problems, and by exploring trade-offs between solution quality and computational cost. This examination of the data is structured around the following three questions, each of which examines a different aspect of overall performance and is answered in its own sub-section.

- How often does each algorithm perform best? (§5.2.1)
- How does the quality of each algorithm's solutions vary? (§5.2.2)
- Are there observable trade-offs between solution quality and computational cost? (§5.2.3)

5.2.1 How often does each algorithm perform best?

Under the two chosen performance measures, an algorithm is the best-performing on a given problem if it either produces the solution with the lowest objective value or consumes the fewest computational resources to arrive at that solution. Fig. 5.4 shows the number of times each algorithm performs best under both measures. Since multiple algorithms can perform identically on a problem (i.e., achieve the same objective value or make the same number of analysis calls), the sum of the numbers in the plots exceeds 100%.

No single algorithm comes close to outperforming all others on every problem. CRS finds the best solution most often, although its population-based approach requires many analysis calls to do so, leading it to never be ranked best in terms of computational cost. COBYLA follows next in terms of solution quality, but is the quickest to converge on 21% of problems. BOBYQA's light computational burden (it is most frequently the algorithm that makes the fewest analysis calls) is offset by its rare convergence on the best solution.

Closer inspection of the data underlying Fig. 5.4, however, shows that algorithms' victories are often narrow. The lightest bridge sometimes weighs only a few kilograms less than the next lightest, and the second-fastest algorithm sometimes requires only a handful more analysis calls to converge. It is therefore useful to relax the measure of 'best' and to count how often an algorithm comes within 5% of the strongest performer. For each of the 474 problems, we determine the set of algorithms a^* such that:

$$a_j^* \in \{a \mid p_{aj} \leq 1.05(p_{min,j})\} \tag{5.1}$$

where p_{aj} is the a^{th} algorithm's (non-normalized) performance on problem j, and $p_{min,j}$

is the best observed performance on problem j. Fig. 5.5 shows how often each algorithm appears in such a set, revealing a more balanced picture of performance.

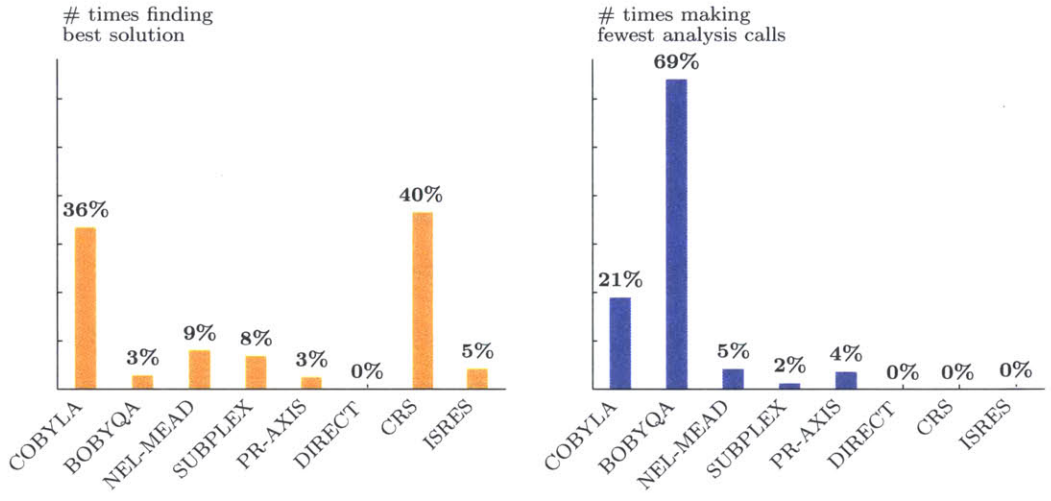


Figure 5.4 – Across all 474 optimization problems, no single algorithm outperforms all others on either measure of algorithm performance—solution quality (objective value) or computational cost (number of analysis calls).

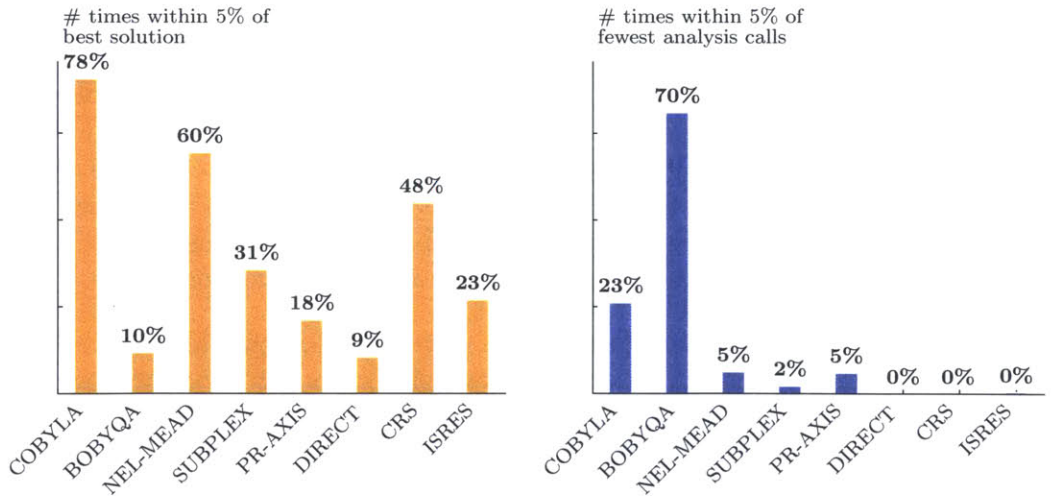


Figure 5.5 – Relaxing the requirement that an algorithm be strictly the best-performing gives a more balanced view of overall performance.

Algorithms that rarely find the highest-quality solution, such as NEL-MEAD or SUB-

PLEX, often find a solution that is almost as good. In 78% of problems, COBYLA finds a solution that is reasonably close to the best observed answer, and is still fairly often among the quickest to converge. The three global algorithms (DIRECT, CRS, and ISRES), of course, make so many more analysis calls than the local ones that they still never come within 5% of being the computationally-lightest.

Counting how often an algorithm is ranked best in this manner is a useful preliminary exploration of the data, showing us that no single algorithm is the best across all problems, and that choosing the correct algorithm for a given design problem is of some benefit. These counts tell us little, however, about the quality of solutions relative to the manually-generated starting points or relative to the best observed solution to a problem—considerations which are of greater interest to engineers, and which are accounted for next.

5.2.2 How does the quality of each algorithm's solutions vary?

Algorithms begin their exploration of each design space from a manually-generated initial solution to a design problem (Chapter 3). A useful and intuitive measure of an algorithm's final solution quality, allowing for comparison across a range of very different problems, is the percentage improvement in the objective function over this starting point. This measure is considered for both the mass minimization problems and the stiffness maximization problems (§3.1). Fig. 5.6 uses boxplots (§5.1) to visualize the range of design improvement achieved by each algorithm on the 158 design problems, where the objective is to minimize mass subject to constraints on displacement and, possibly, on stress. The remaining 316 problems seek to minimize the exceedance of allowable displacement (and, hence, to maximize stiffness) subject to constraints on mass; the ranges of design improvement for these problems are shown in Fig. 5.7[2]. Both sets of boxplots show only the range $[-100\%, 100\%]$.

Fig. 5.6 shows a relatively consistent pattern of design improvement on the mass minimization problems. Most algorithms reduce material usage by, on average, between 10% and 30%. As suggested by the frequency with which it produces a near-best solution (Fig. 5.5), COBYLA produces the best range of mass reduction across the 158 problems. On the other hand, three algorithms—PR-AXIS, CRS, and ISRES—produce, on average, solutions that are no lighter than the starting designs, although each one's range of solutions does include some improved designs. The remaining four algorithms are much more likely to improve designs, although they do so to differing extents, indicated by the wide ranges of solution quality. Closer inspection of the data (presented later in §5.3) show that

[2]The goal of the stiffness problems is to minimize $|\Delta(\vec{x})|_{\max} - \Delta_{\text{limit}}$ (Eq. 3.2). This quantity can take on negative values, however, making percentage improvement calculations meaningless. We instead use $|\Delta(\vec{x})|_{\max}$, the maximum deck displacement magnitude, to measure starting point and solution quality when plotting Fig. 5.7.

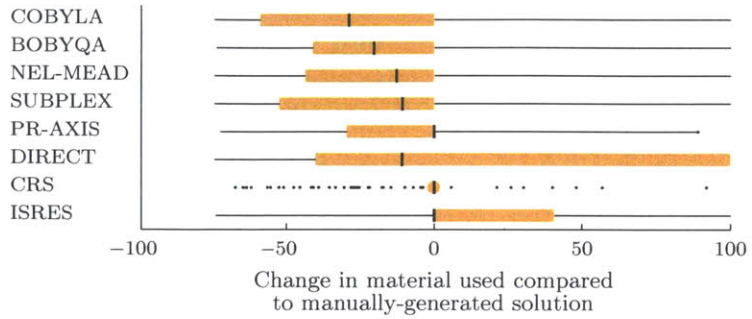


Figure 5.6 – Overall, algorithms show a fairly consistent pattern of improvement over the manually-generated starting solutions to the 158 mass minimization problems, although all algorithms produce some solutions that are worse than the starting points. Some algorithms show a more consistent pattern of improvement than others, and almost all of them exhibit wide ranges of solution quality.

intra-algorithm variation in solution quality across types of problems is also significant. Algorithms do not achieve their best success on the same subset of the mass minimization problems; rather, an algorithm's performance depends strongly on the problem at hand.[3].

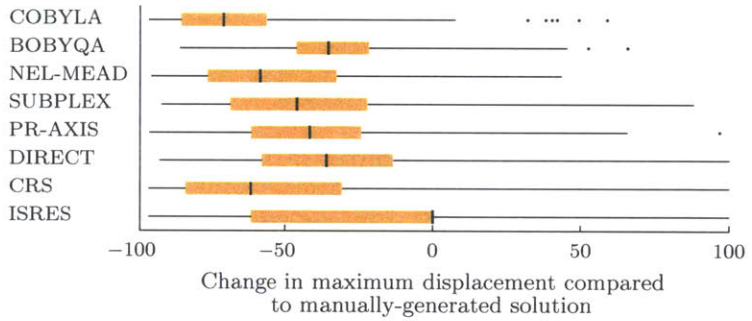


Figure 5.7 – Across the 316 stiffness maximization problems, algorithms consistently find solutions with lower maximum displacement values than the manually-generated starting designs. The wide ranges of improvement for each algorithm differ from the mass minimization problems (Fig. 5.6).

Fig. 5.7 shows a more consistent pattern of design improvement on the 316 stiffness optimization problems. Since most of the manually-generated starting points use less than the allowable amount of material (i.e., since the upper-bound constraints on mass were

[3]These design improvement plots show many cases where an algorithm produces a solution that is worse than the starting point. In practice, an algorithm encountering this difficulty would simply return the starting point with a comment that it was unable to improve the design. As part of the overall assessment of algorithm quality, we opt for the lee favorable representation shown here.

not active at the starting point), this is to be expected. COBYLA and CRS produce better ranges of stiffness solutions than the other algorithms, although both are often outperformed. The high frequency with which CRS produces the best solution (Fig. 5.4) is attributable to its strong performance on these 316 stiffness problems, since its performance on the mass minimization problems is so poor (Fig. 5.6). The broad message of Fig. 5.7 is the same as for the mass minimization problems: a range of representative optimization algorithms demonstrate a consistent pattern of improvement across many design problems, but inter- and intra-algorithm variation in solution quality is substantial.

Although useful in evaluating design improvement and, hence, solution quality, these visualizations are limited by the fact that they compare solution quality only to the manually-generated starting points; different starting points produced by different engineers could alter the results significantly. In theory, a comparison of solutions to the true optima is desirable. For the simulation-based non-benchmark problems studied here, however, these true optima are unknown. The remainder of this section instead uses the best among all determined solutions (including the starting points) to a given problem as a substitute. Each solution to a problem j in the data is then assigned a gain value $\mathcal{G}_j$ according to Eq. 5.2.

$$\mathcal{G}_j(y_j(a)) = 1 - 2^{-1.5}\left(\frac{y_j(a) - y_{j,\min}}{y_{j,\min}}\right)^{1.5}; \qquad \mathcal{G}_j \leq 1 \tag{5.2}$$

where $y_j(a)$ is the objective value attained by algorithm a on the j^{th} optimization problem, and $y_{j,\min}$ is the minimum observed objective value (i.e., the best solution) for the j^{th} problem. $y_{j,\min}$ substitutes for the true, unknown optimum solution to the problem. $\mathcal{G}_j(y_j(a))$ has a value of 1.0 when algorithms a's solution is the best observed (i.e., $y_j(a) = y_{j,min}$), and decreases with a growing rate of decline as solutions become progressively worse than the best. Solutions that are 200% worse than best have a gain of zero.

Fig. 5.8 shows the summation of each algorithm's gain values across all 474 problems—a single measure of overall solution quality. Any solutions over 200% worse than the best solution (which have negative gain values) are discarded before taking this summation. Engineers can usually approximate the optimal solution's objective value in advance, using, for example, expected trends in material usage such as those shown in Fig. 4.1. Solutions that are 200% worse than the optimum should, therefore, be easily identified and discarded by engineers as sub-optimal, justifying their omission from the cumulative gain measure. The number of these discards is, however, noted for each algorithm.

Fig. 5.8 shows that COBYLA produces—in the sense defined by Eq. 5.2 and by the decision to discard designs over 200% worse than the lightest or stiffest solutions—the best overall set of solutions to the design problems. This is not to say that COBYLA is always best; Fig. 5.4 and the deeper examinations of the data in §5.3 prove otherwise.

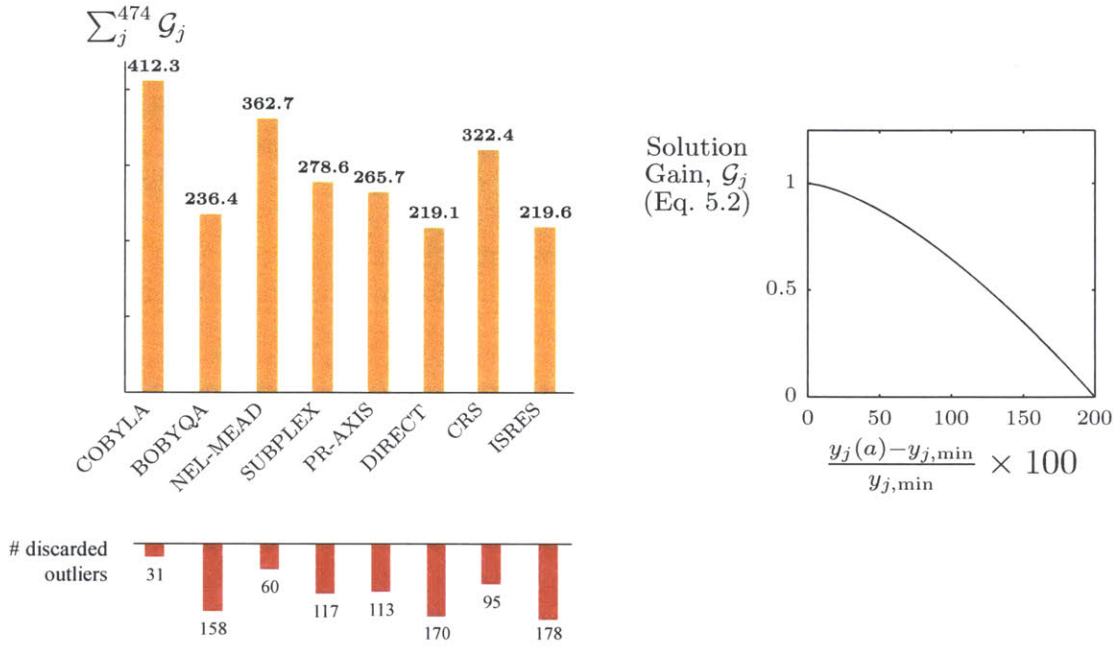
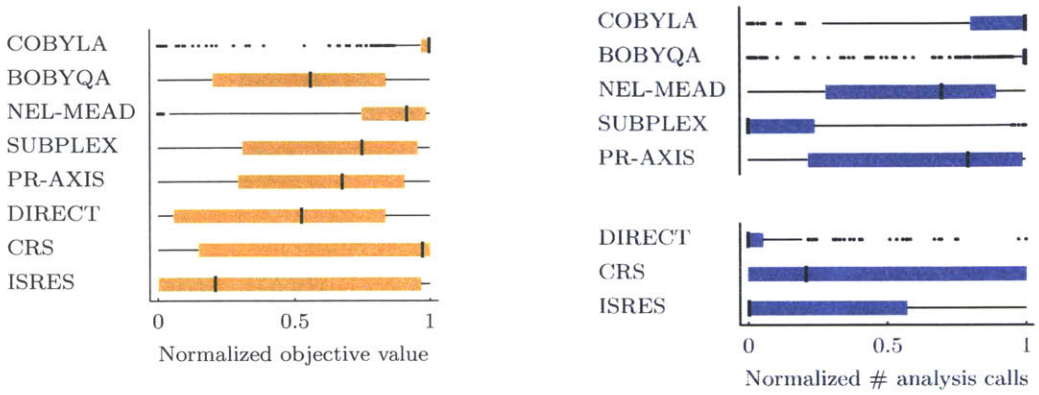


Figure 5.8 – Summing the range of gain values (Eq. 5.2, plotted on the figure's right) across the 474 design problems gives a single measure of overall solution quality for all algorithms, and demonstrates the importance of looking beyond how often each algorithm is ranked best (Fig. 5.4). Each algorithm finds, with varying frequency, a solution which is over 200% worse than the best observed solution; these outliers are discarded before calculating $\sum_j^{474} \mathcal{G}_j$.

CRS, the algorithm which most frequently produces the best solution, has an overall gain measure which is significantly lower than COBYLA's and NEL-MEAD's. CRS's score is closer to that of PR-AXIS, an algorithm which finds the best solution less than 10% as often as CRS does. This demonstrates the problem with simply considering the number of times an algorithm is best; an algorithm's performance on those problems where it is not the best—the extent of its downside—is an important part of the picture.

5.2.3 Are there observable trade-offs between solution quality and computational cost?

The previous section considered distributions only in algorithms' solution quality, without regard to the computational cost associated with producing those solutions. As a final component in the analysis of overall performance, this section concurrently explores variations in solution quality and computational cost.



(a) (b)

Figure 5.9 – (a) Solution quality. The overall range of normalized solution quality (Eq. 3.8) varies widely across all algorithms other than NEL-MEAD and COBYLA; any algorithm may produce relatively good or relatively poor solutions, depending on the problem at hand. **(b) Computational cost.** Normalized separately among local and global algorithms, the computational cost data vary widely. In both cases, higher numbers indicate better performance.

Using the normalization strategy presented in Chapter 3 (Eq. 3.8), which assigns values between 0 and 1 to the algorithms for each problem, Fig. 5.9 provides some insight into overall trends. Most algorithms have a wide range of normalized objective values (solution quality) and normalized number of analysis calls (computational cost)[4].

Among the local optimization algorithms, which terminate when the objective value converges within 1×10^{-4} and which are much computationally lighter than the three global algorithms, BOBYQA is most frequently the fastest to converge across all problems, followed by COBYLA. Their successive approximations of the design space likely allow them to progress quickly. The global algorithms—DIRECT, CRS, and ISRES—make as many calls as possible within their imposed six-hour time limit. DIRECT's and ISRES's lower normalized values actually indicate that they make *more* analysis calls in a fixed period and are, in a narrow sense, more efficient. Despite considering more design iterations, however, they produce solutions of lower average quality than CRS.

Fig. 5.10 plots each algorithm's normalized objective value against its normalized number

[4]As discussed in §3.2.2, normalized computational cost is determined separately for local and global algorithms. Although we do not explicitly examine computation times, most local algorithms take approximately 5 to 15 minutes to converge—at least two orders of magnitude less than the global algorithms, which always take 6 hours. Some local algorithms converge in less than a minute, however, and others (especially SUBPLEX) can take over an hour.

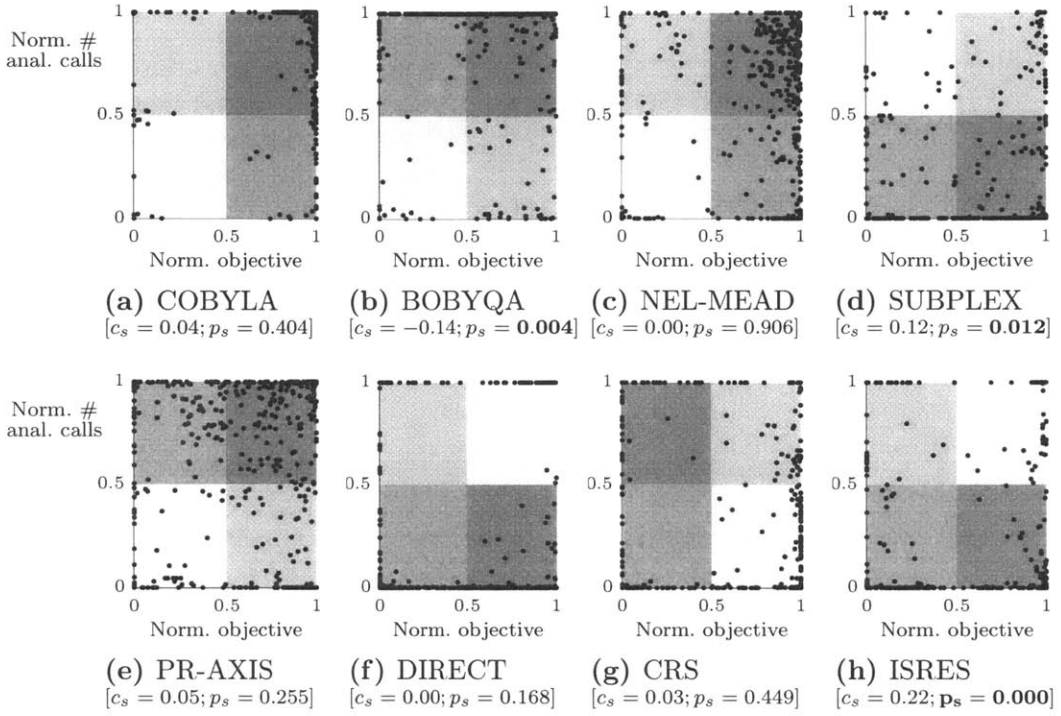


Figure 5.10 – Each plot shows a single algorithm's normalized (Eq. 3.8) solution quality achieved and the computational cost incurred on the 474 problems. Darker-shaded quadrants contain more points, and captions note the Spearman Rank Correlation coefficients between the two measures, c_s, and their associated p-values, p_s (§5.1).

of analysis calls for all 474 design problems; the background of each plot is split into four quadrants, with darker colors assigned to quadrants containing more points. Although the measures are not strongly correlated for all algorithms ($p_s \leq 0.05$ for only three of them), certain trends do emerge.

All five local algorithms (COBYLA through PR-AXIS) apart from SUBPLEX rarely take a long time to reach poor solutions (their lower left quadrants contain relatively points). In cases where they make many analysis calls, they arrive at relatively good solutions. SUBPLEX, by contrast, on those occasions where it takes longer to converge, is more likely than the other other algorithms to do so on a poor-quality design. The top left quadrants of BOBYQA's and PR-AXIS's plots shows that both algorithms often converge quickly to a bad solution, suggesting a propensity to getting stuck early in local optima.

Of the global algorithms (DIRECT, CRS, and ISRES), only ISRES shows a statistically significant association (indicated by a p_s value less than 0.05); its solution quality increases as it makes more analysis software calls. DIRECT and ISRES, as evidenced by the large number of points in their plots' bottom right-hand quadrants, are more likely to produce

good solutions when they make relatively many analysis calls. CRS demonstrates the opposite trend, achieving better results on problems where it makes fewer analysis calls than the other global algorithms.

Summary

This section, by demonstrating how a set of algorithms performs on a wide range of realistic problems, answers the dissertation's first research question and takes a step towards resolving a key identified limitation of the optimization literature—the insufficient demonstration of structural optimization's true potential. By running all algorithms 'off-the-shelf', with their various tuning parameters set to default values (Chapter 3), and by presenting the entire range of resulting outcomes—including instances where algorithms fail to improve on the initial design—the data offer a clearer, more honest picture of optimization's promising, but variable, performance than the contemporary literature.

§5.2.2 demonstrates the importance of looking beyond counts of how often an algorithm is best to the quality of its solutions. Although CRS most frequently produces the best solution, for example, it exhibits significant downside on those occasions where it does not do so. COBYLA produces the best overall range of solutions under the defined evaluation metrics, although there are many occasions where it is outperformed. §5.2.3 considers overall computational cost trends in addition to those in solution quality, further demonstrating the complexity of the problem.

The section also reaffirms the findings of the short performance variation study in the dissertation's first chapter (§1.2), although now with far more data across a broader range of structural types. There is much variation in algorithmic performance across different problems, and no single algorithm outperforms all others. Algorithms that frequently find the best solution are often the most computationally expensive, and often have significant downsides in the cases where they don't find the best answer.

All of this confirms our initial assertions regarding the importance of the algorithm selection problem, and implicitly proposes a strategy for comprehensive evaluation of algorithms on a wide range of realistic problems. As a first step towards guiding algorithm selection, the remainder of this chapter delves deeper into the data in search of relationships between characteristic features of design problems and the performance of each algorithm in solving those problems.

5.3 Association between problem features and algorithm performance

The second question posed in the dissertation's introduction is: *"Do correlations exist between problem features and algorithm performance?"* This section explores the relationship between performance—both solution quality and computational cost—and key features of the structural optimization problems. The investigation is organized around the following questions, and §5.3.6 concludes by making broad recommendations about when each algorithm should be used.

- Does algorithm performance vary with the nature of the design variables? (§5.3.1)
- Does algorithm performance vary with the type of objective function? (§5.3.2)
- How does algorithm performance vary with the number of design variables? (§5.3.3)
- Does the degree of constraint affect algorithm performance? (§5.3.4)
- Do algorithms perform differently across different structural types? (§5.3.5)

5.3.1 Does algorithm performance vary with the nature of the design variables?

Every optimization problem in the study can be classified as either a sizing problem or a joint sizing and geometry (hereafter referred to as geometry) problem, depending on whether the design variables control overall structural geometry. Fig. 5.11 shows the

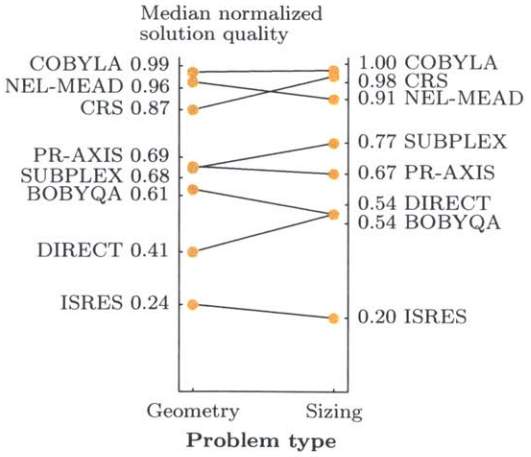
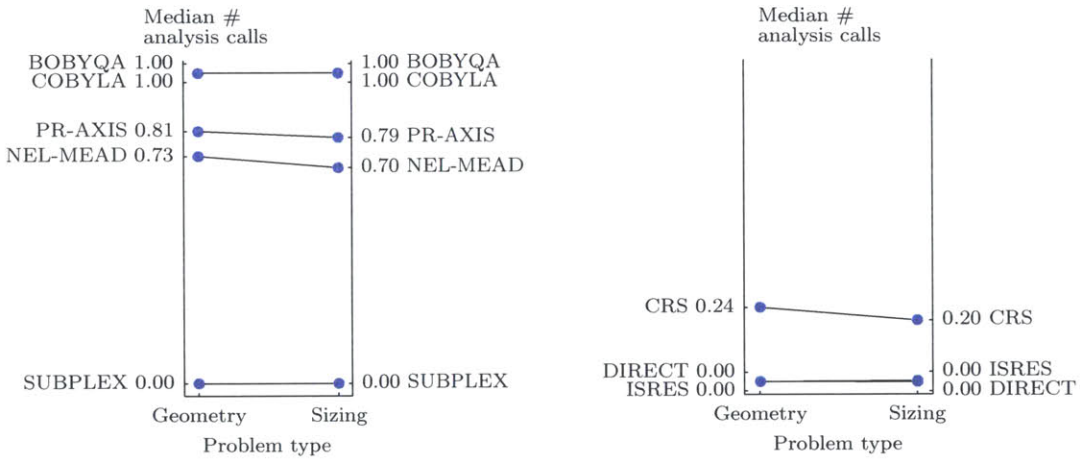


Figure 5.11 – Four algorithms have a lower median normalized objective value (i.e., their median solution decreases in quality) on sizing problems (Eq. 3.8).

algorithms' median normalized objective value (a single measure of overall solution quality, calculated as the median of the data resulting from Eq. 3.8) for problems of each type. Higher numbers indicate better solution quality.

Four algorithms—COBYLA, SUBPLEX, DIRECT and CRS—produce relatively better solutions, on average, when solving sizing problems. Among these four, however, the Kruskal-Wallis test gives $p_{kw} \leq 0.05$ only for COBYLA and DIRECT; the difference in performance for the other two algorithms is deemed statistically insignificant. NEL-MEAD and BOBYQA both generate significantly better solutions for geometry problems ($p_{kw} = 0.014$ and 0.029, respectively).

The figure shows only median performance values, and is therefore a low-information representation of the underlying data ranges. §C.2 in Appendix C plots the full ranges behind these median data for this and for the following sections[5]. These additional plots enhance understanding of the variation in performance. In this case, they show that NEL-MEAD, PR-AXIS, and CRS, have significantly more downside for sizing problems than they do for geometry problems, while the opposite is true of COBYLA (Fig. C.3).



(a) Local algorithms **(b)** Global algorithms

Figure 5.12 – The median normalized number of simulation calls (a measure of average computational cost) does not vary greatly across these problem types. Fig. C.3 shows the ranges of values behind these medians. Higher numbers indicate fewer calls to the analysis software (better computational cost), following the normalization strategy of Eq. 3.8.

Fig. 5.12 shows that the nature of the design variables is not an important predictor of computational cost—these plots, and hypothesis testing ($p_{kw} > 0.05$ in all cases), show insufficient evidence to reject the null hypothesis that problem type and normalized number of analysis calls are not associated.

[5]Distributions around median values, however, are considered here as part of the Kruskal-Wallis hypothesis testing.

In summary, the quality of solutions produced by algorithms varies somewhat with the nature of design variables, but the incurred computational cost does not. We should take care to remember, of course, that any identified associations tell us little about the true causes of performance variation; changes in any number of other problem features may be responsible for the observed effects.

5.3.2 Does algorithm performance vary with the type of objective function?

This analysis is similar to the previous one, except problems are now binarily classified by the nature of the objective function. As discussed in Chapter 3, all problems seek either the lightest or the stiffest solution; their respective objective types are denoted 'mass' or 'stiffness' here.

The variation in median normalized performance measures is more dramatic than that observed in the previous section. Fig. 5.13 shows that all algorithms apart from CRS have worse median normalized outcomes for stiffness problems than they do for mass problems. Kruskal-Wallis tests show sufficient evidence to support the visual trends in median objective value; the hypothesis of different average values is rejected only for COBYLA, PR-AXIS, and ISRES[6].

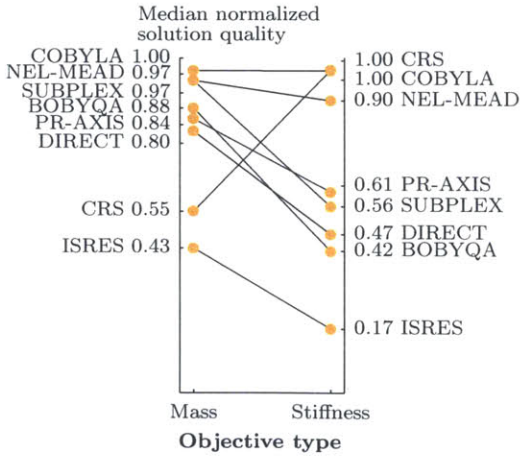
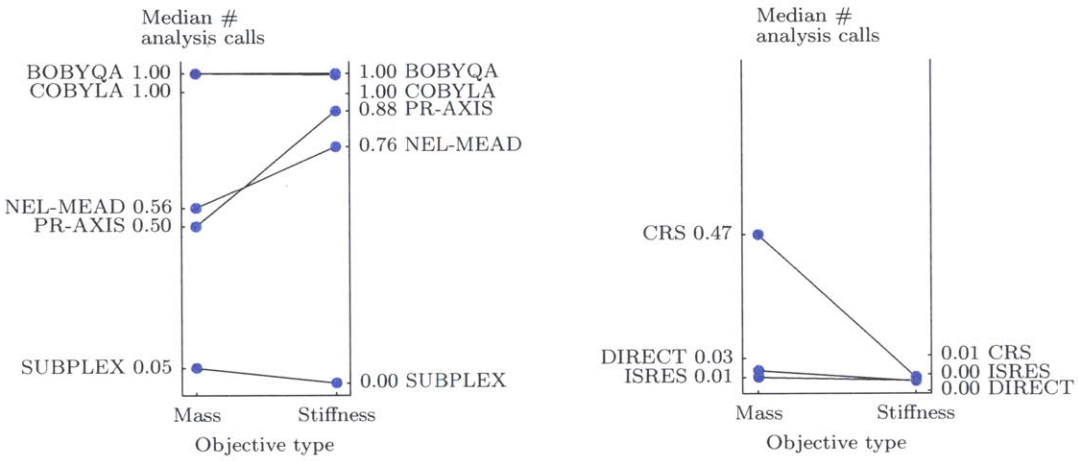


Figure 5.13 – All algorithms other than CRS produce worse median solutions for stiffness maximization problems than for mass minimization ones.

Several algorithms have significantly different median computational costs across these problem categories (Fig. 5.14). Global algorithms make relatively more analysis software calls in their six hour time limit on stiffness problems than on mass problems. The analysis

[6]Although the differences in PR-AXIS's and ISRES's median normalized objective values are large, the sizeable variation in the underlying data prevents the statistical test from showing a significant effect (Appendix C, Fig. C.4).

software-based objective and constraint functions in the stiffness problems are, in general, computationally lighter than those on the mass problems, especially when problems rely on geometrically nonlinear analyses which tend to converge quicker as displacements decrease. This allows the global algorithms to make relatively more analysis calls in a fixed period of time. The local algorithms (except for SUBPLEX) converge more quickly on stiffness problems.



(a) Local algorithms **(b)** Global algorithms

Figure 5.14 – All local algorithms other than SUBPLEX show a relative decrease in incurred computational cost. The less expensive structural analyses used in stiffness problems allows the global algorithms to make relatively more analysis software calls. The benefit in solution quality is realized only by CRS, however (Fig. 5.13).

In summary, the type of objective function is an important predictor of solution quality for five of the eight algorithms, and of computational cost for many algorithms. As with the previous section, the ranges of performance values behind these medians are shown in Appendix C (Fig. C.4).

5.3.3 How does algorithm performance vary with the number of design variables?

The number of design variables is often used in optimization texts and studies as a characteristic for selecting algorithms, motivating an investigation of its role here even though it is not one of the primary problem features (§3.1).

The number of design variables per problem, however, does not vary greatly in this study; observed values instead fall into a small, finite set. Recognizing this, Figs. 5.15 and C.1

eschew the standard scatterplot used to examine correlations of this nature, and instead plot only an algorithm's median performance across all problems with a given number of design variables, with error bars extending to the 25^{th} and 75^{th} quantiles. The Spearman correlation coefficients (c_s) and p_s-values shown in the sub-captions are calculated over the entire range of each algorithm's performance measures—not over the median values alone.

The plots and Spearman values show statistically significant, although far from perfectly monotone, correlations. As the number of design variables increases, all the local algorithms, apart from NEL-MEAD and SUBPLEX ($p_s \leq 0.05$), produce better solutions. Two of the global algorithms—DIRECT and CRS—by contrast, produce progressively worse solutions when dealing with more design variables. Given the exhaustive manner in which the global algorithms search the design space, and their known difficulty in maintaining performance as problem dimensionality increases, this is not surprising.

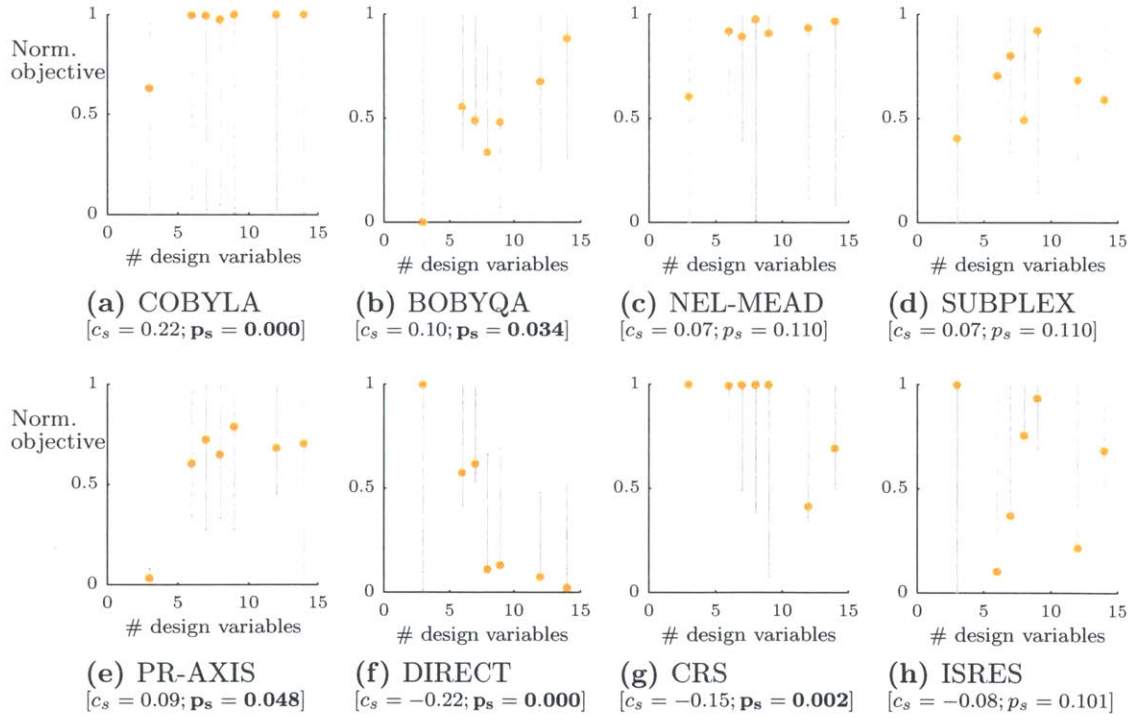


Figure 5.15 – All algorithms, apart from NEL-MEAD, SUBPLEX, and ISRES, show a statistically significant association between median solution quality and the number of design variables. The local algorithms (especially COBYLA, BOBQYA, and PR-AXIS) perform relatively better as the number of design variables increases; the global algorithms perform relatively worse. c_s values indicate the direction and the strength of rank correlation, and p_s tests the hypothesis that c_s is significantly different from zero.

Correlations between normalized computational cost and the number of design variables are not as strong; they are significant for only two of the eight algorithms. The relevant visualizations and statistical measures are shown in Appendix C, Fig. C.1.

5.3.4 Does the degree of constraint affect algorithm performance?

The degree of constraint is another mathematical feature likely to play an important role in determining algorithm performance. It is, however, a difficult attribute to define concretely, since different constraints may have quite distinct effects on the mathematical nature of the design space. To address this, we propose two measures of a problem's degree of constraint, and look for algorithm performance trends across both. These measures are: the integer number of constraints, without regard to their type, and the categories of constraint present.

(a) The integer number of constraints, without regard to their type

Figs. 5.16 and 5.17 plot normalized algorithm performance against the number of constraints for each algorithm; captions note the Spearman rank correlation coefficients and p-values. As in the previous section, the figures show median performance for a given number of constraints, with bars extending to the 25^{th} and 75^{th} percentiles of performance ranges.

Considering the relationship between the integer number of constraints and algorithm performance, the five local algorithms appear to find relatively better solutions for less-constrained problems (Fig. 5.16). Spearman rank correlation testing, however, rejects these associations for all algorithms other than SUBPLEX (which generally performs better on more-constrained problems. The global algorithms, on the other hand, have negative c_s values, suggesting that their solutions become worse as problems become more constrained. Again, p_s values reject this hypothesis for DIRECT and CRS; only ISRES ($c_s = -0.23$, $p_s = 0.002$) has a supportable trend of worse solutions for constrained problems.

Strong associations emerge more often in the plots of normalized number of analysis calls against the number of constraints (Fig. 5.17), as confirmed by p_s-values of less than 0.05 for all algorithms except BOBYQA and NEL-MEAD.

The local algorithms—especially COBYLA and PR-AXIS—require relatively more analysis software calls to converge as problems become increasingly constrained, perhaps due to increased problem complexity. (SUBPLEX, however, shows the opposite trend.) The

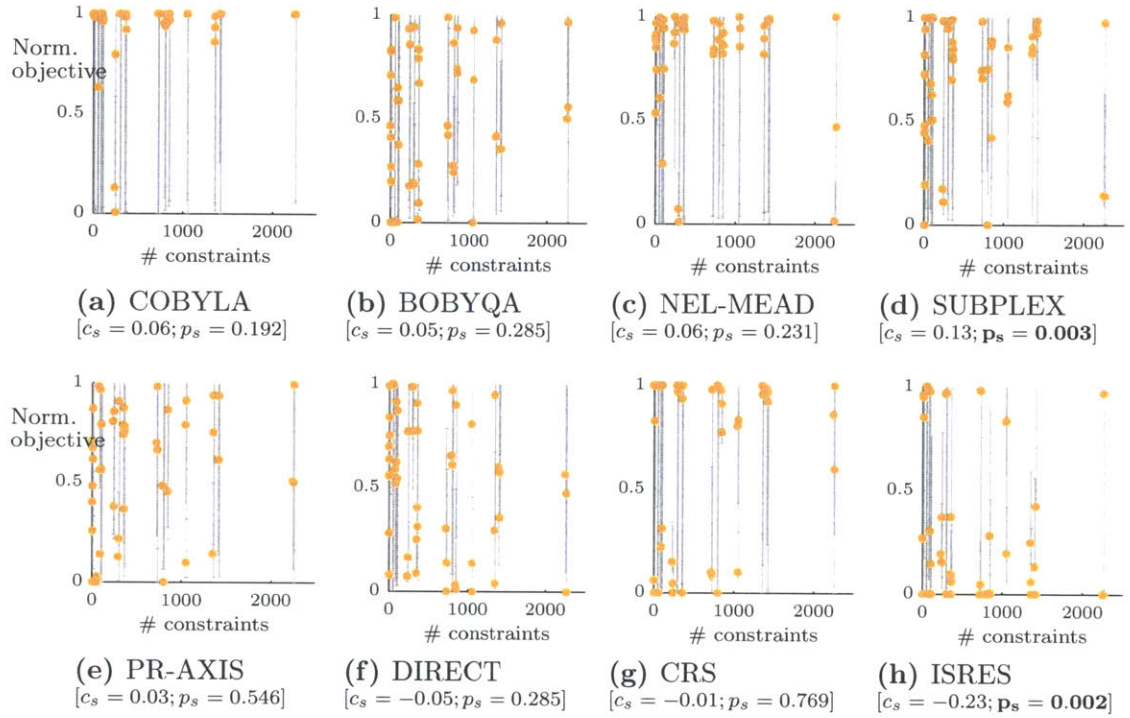


Figure 5.16 – The number of constraints is not significantly correlated with normalized objective value for any algorithm except SUBPLEX (whose solution quality improves with the number of constraints) and ISRES (which displays the opposite trend).

global algorithms all make fewer analysis calls in a fixed time period as the degree of constraint increases. The most likely cause of this is the increased computational cost per design iteration of evaluating more constraints.

(b) The types of constraint present

The experimental design contains three types of constraint, defined by the physical property being constrained—stiffness (S), displacement (D), and mass (M). The experimental design constrains each problem with one of six possible combinations of these types—D only; D and S; M only; M and S; M and D; and M, S, and D. Figs. 5.18 and 5.19 examine how algorithm performance varies across problems constrained in each of these six ways. When evaluating performance trends, one should note that problems with the categories D and $D+S$ (plotted on the extreme left of Figs. 5.18 and 5.19) have minimum mass as their objective, and problems in the other four constraint categories have maximum stiffness as their objective. These plots, therefore, simultaneously show performance variation across constraint categories and across objective types.

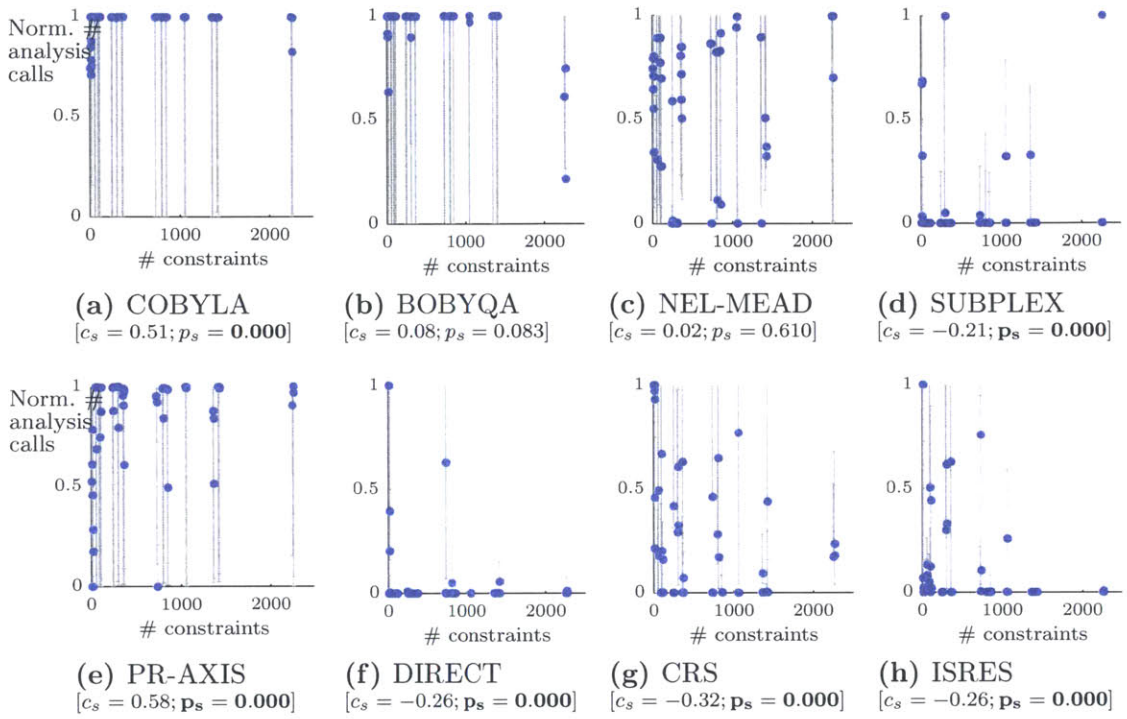


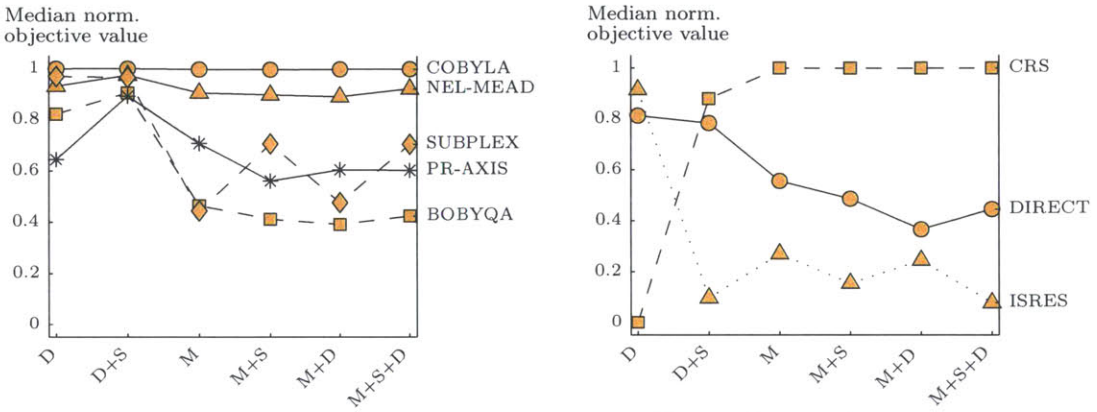
Figure 5.17 – The number of constraints correlates with normalized computational cost for all algorithms except BOBYQA and NEL-MEAD. Local algorithms make more analysis calls on increasingly-constrained problems, and global algorithms can make fewer analysis calls in a fixed period as the number of constraints and, hence, the computational cost of a design iteration increases.

The variation in normalized objective attained across constraint categories is statistically significant ($p_{kw} \leq 0.05$) for all algorithms apart from COBYLA, which performs well, on average, regardless of the categories of constraint. SUBPLEX is the only algorithm that shows dramatically better performance as constraint categories are added, and ISRES is the only one to strongly show the opposite trend; both of these interpretations match our expectations given the similar associations between the number of constraints and these algorithms' performance.

The plots of median computational cost across constraint categories show statistically significant variation for all algorithms (Fig. 5.18), largely due to low variation around the observed medians (shown in Fig. C.5's plots of the underlying data ranges).

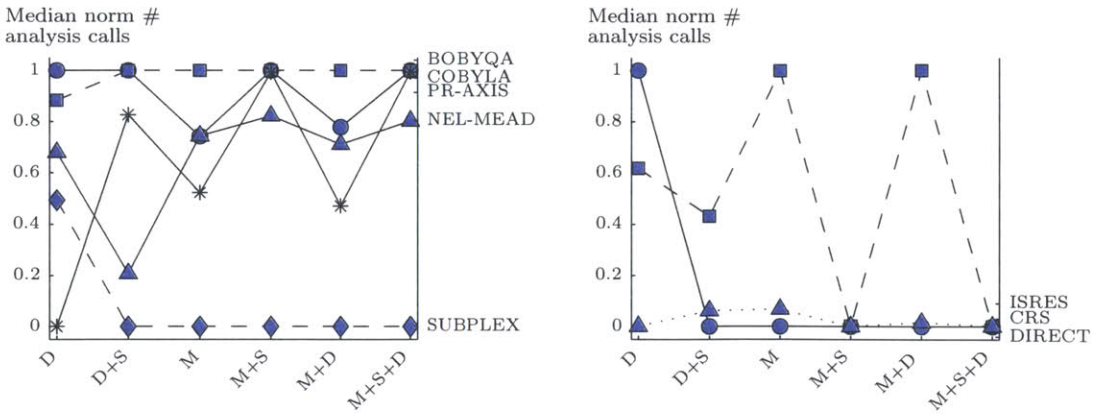
In summary, the degree of constraint—whether measured as the integer number of constraints or the constraint categories present—is a more significant predictor of computational cost than of solution quality. All algorithms, apart from CRS and ISRES (although the latter's trends are not always statistically significant), require more analysis calls to

converge on highly-constrained problems. The higher computational cost is likely due to the complexity of increasingly-constrained design spaces.



(a) Local algorithms **(b)** Global algorithms

Figure 5.18 – Comparing median normalized objective values across problems categorized by the types of constraints present shows significant performance variation for some algorithms. Others perform consistently regardless of the constraint types present. Figs. C.6 and C.5 expose the data ranges behind these medians.



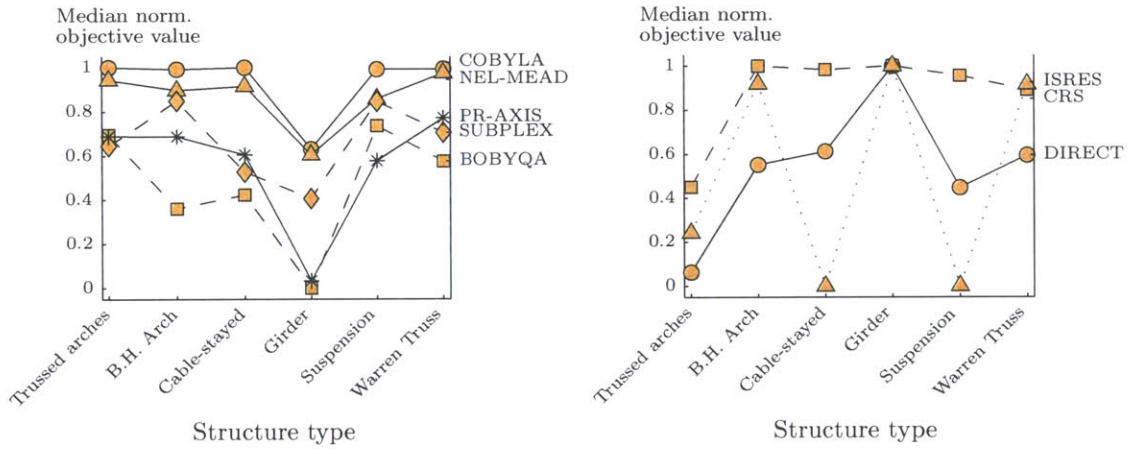
(a) Local algorithms **(b)** Global algorithms

Figure 5.19 – Comparing median normalized analysis software calls across problems categorized by the types of constraints present shows significant performance variation for some algorithms. Others perform consistently, regardless of the constraint types present. Figs. C.6 and C.5 expose the data ranges behind these median values.

5.3.5 Do algorithms perform differently across different structural types?

Each optimization problem is based on a structure which falls into one of six categories—trussed arches, basket-handle arches, cable-stayed bridges, girder bridges, suspension bridges, and Warren trusses. This section explores algorithms' solution quality on problems across these six structural types. Fig. 5.20 plots the algorithms' median normalized solution quality. Appendix C.1 does the same for performance measured in terms of computational cost (Fig. C.2), and Appendix C.2 show data ranges behind these medians (Figs. C.7 and C.8).

Solution quality varies widely across structural types, with only CRS, NEL-MEAD, and COBYLA performing relatively consistently (Fig. 5.20). The median number of analysis calls made by algorithms in solving problems of each structural type varies less (Fig. C.2). $p_{kw} \leq 0.05$ for all algorithms on both performance measures, indicating that structural type is strongly associated with an algorithm's average performance.



(a) Local algorithms

(b) Global algorithms

Figure 5.20 – The median normalized solution quality for each algorithm varies across the six structural types, and all algorithms have at least one type for which their median performance is unusually poor. COBYLA, NEL-MEAD, and CRS are the most consistent. (See Fig. C.7 for numerical ranges behind these medians.)

The major performance trends, arranged by structural type, are:

Trussed arches There is a dramatic difference between the local and the global algorithms on the trussed arch problems. The global algorithms, especially DIRECT and ISRES, find poor solutions on average—much worse than even the worst of the

local algorithms. In light of the trends observed in previous sections, this makes sense. The trussed arches have the greatest number of design variables and—due to their large numbers of elements—the most stress constraints of any structural type. Global algorithms appear to perform poorly on problems with many variables and high-dimensional, extensive search spaces; their population-based approach requires very many analysis software calls to successfully search these larger design spaces, which they are unable to do in the allotted six hours.

Basket-handle arches The basket-handle arch problems have relatively few design variables and reasonably well-behaved design spaces. The fairly consistent solution quality across many algorithms is therefore unsurprising[7]. Although DIRECT appears to encounter difficulty, the other two global algorithms perform well in these relatively small design spaces, where their population-based approaches are effective.

Cable-stayed bridges A primary characteristic of the cable-stayed bridges is the computationally-expensive, geometrically nonlinear structural analyses that must be run for each design iteration. This creates difficulty for the global algorithms, which accordingly make fewer analysis software calls in their allotted time and attain lower-quality solutions. This is particularly true of ISRES.

Girder bridges The girder bridges are an unusual group of problems in terms of their performance trends. Local algorithms that otherwise generate good solutions, including COBYLA and NEL-MEAD, fail to do so here. Instead, the global algorithms all achieve the best median solutions, although they require very many analysis calls to produce them. The girder bridges, despite their physical simplicity, have relatively complex design spaces with many local optima. These local optima are the most likely cause of difficulty for the local algorithms, but they pose less of a problem for the global algorithms. Given the small size of the design space (the problems have only three sizing variables) and light computational load during analysis, the global algorithms can explore very many solutions in their six hour time limit.

Suspension bridges As with cable-stayed bridge problems, ISRES is by far the worst performer here. CRS and COBYLA produce the best average results, although CRS requires many more analysis calls to do so. SUBPLEX and NEL-MEAD produce a range of designs likely to be acceptable in many scenarios, and consume relatively few resources in doing so.

Warren trusses The overall variation in the range of solution qualities is very large for the 118 Warren truss-based optimization problems; only NEL-MEAD and COBYLA have reasonably restricted ranges of solution quality. Although CRS has the best

[7]BOBYQA's and PR-AXIS's poor solution quality is likely due to a propensity to converge quickly on low-quality local optima (Fig. C.2). SUBPLEX, which takes much longer to converge, produces good solutions on average

median solution quality, it shows significant downside by otherwise producing sub-optimal designs.

In summary, there is a strong association between structural type and the quality of solutions; every algorithm's median objective values vary significantly. Although some algorithms are more consistent than others, they all find unusually poor solutions within at least one of the structural types. The median number of analysis calls is less variable (Fig. C.2), but the investigations show that structural type is still an important predictor of computational cost.

5.3.6 Summary and discussion of observed trends

This section set out to answer the dissertation's second research question: *"Do correlations exist between problem features and algorithm performance?"* In doing so, the previous five sections present a set of important trends in algorithm performance, showing that some problem features are more closely associated with performance than others.

Although the nature of the design variables shows some association with algorithms' solution quality, the type of objective function turns out to be a much more significant high-level predictor of both solution quality and computational cost (§5.3.1 and §5.3.2). Algorithm performance also varies significantly across structural type, with even the best-performing algorithms exhibiting markedly poor performance on at least one structural type (§5.3.5).

Increasing numbers of design variables are generally associated with relatively better solutions for local algorithms, and with relatively worse solutions for global algorithms (§5.3.3). The problems' degree of constraint does not show such close association with solution quality, although increasingly-constrained problems require all but one of the local algorithms to consume more computational resources, and limit the number of analysis calls which global algorithms can make (§5.3.4).

Overall, however, it is clear that there are no independent, linear correlations between these important features and algorithm performance. Although the trends stated in the chapter are supported by rigorous statistical testing, the somewhat low correlation factors and the large variations around many of the median values (Appendix C.2) means that any recommendations based on these trends are approximate. In reality, the relationships between features of design problems and the performance of algorithms are likely to be nonlinear, complex, and interdependent. Manually exploring all possible nonlinearities and combinations of features in the style adopted here is infeasible, and the problem should be addressed with more complex predictive methods. This is the approach taken in Chapter 6.

Nevertheless, the data investigations in this section are an important first step to understanding the trends in the data and to addressing the algorithm selection problem. We conclude the chapter by summarizing the major trends in performance for each algorithm; these summaries can be used to broadly guide engineers' choice of algorithms in practice.

1. COBYLA

Constrained Optimization by Linear Approximation [Powell, 1994]

COBYLA fits linear approximations via a simplex of size $n+1$ to the n-dimensional design space, optimizes over these approximations within a trust region, and then successively refits the approximation based on the observed error. This strategy allows the algorithm to progress quickly through the design space, but may lead it to converge on local optima in non-convex design spaces.

It is the most consistent performer in solving the optimization problems in this study, finding the best solution to over a third of the problems, and a solution within 5% of the best answer in 78% of cases. It is less frequently the quickest to converge, but its number of analysis software calls is close to being the lowest in roughly a quarter of cases (Figs. 5.4 and 5.5).

That said, closer inspection of the data demonstrates that COBYLA is not always the best choice. It is outperformed on several classes of problem, most clearly on the girder bridges, whose design spaces contain many poor-quality, locally-optimal solutions. Its performance in solving the problems based on Warren truss and suspension bridges is also variable (Figs. C.7 and C.8). On those problems where COBYLA performs poorly, it often makes relatively few analysis calls. This may indicate a tendency to quickly converge on locally optimal, but globally sub-optimal, answers (Fig. 5.10a).

2. BOBYQA

Bounded Optimization by Quadratic Approximation [Powell, 2009]

BOBYQA has a similar strategy to COBYLA, a key difference being the quadratic nature of its successive design space approximations. These quadratic approximations can, of course, lead to large approximation errors in cases where the design space is not twice-differentiable.

Its most notable performance feature is its frequent light use of computational resources; it makes the fewest analysis calls among all algorithms in 70% of cases. The price of choosing an algorithm with such fast convergence times for these problems, however, is

the poor quality of its solutions. BOBYQA's design is within 5% of the best only 10% of the time (Figs. 5.4 and 5.5). Indeed, BOBYQA very frequently converges quickly on local optima which represent poor solutions to the posed problems (Fig. 5.10b). BOBYQA is, in a a sense, a very high-risk choice. On certain problems, such as the trussed arches or suspension bridges, it delivers high-quality outcomes while making very few analysis calls. Much more often, however, it produces poor ones.

The difference in BOBYQA's solution quality distribution is significant when comparing mass minimization to stiffness maximization and geometry optimization to sizing; the algorithm's range of outcomes is significantly better for mass minimization problems and for geometry optimization (Figs. 5.11 and 5.13). Despite these occasional positive trends, BOBYQA does not dominate other algorithms on any of the groupings of problems examined in the chapter.

3. NEL-MEAD

Nelder-Mead Simplex [Nelder and Mead, 1965]

NEL-MEAD progresses by moving a simplex of size $n+1$ through the n-dimensional design space via a series of reflections, expansions, and contractions. This is a slower process than the previous two algorithms, but one that is less likely to fall victim to sub-optimal local peaks in non-convex design spaces.

Although rarely the very best algorithm in terms of solution quality or computational cost, NEL-MEAD performs consistently well across problems; its overall range of solution quality is, arguably, second only to COBYLA's and it usually makes relatively few analysis calls (Figs. 5.8 and 5.9). It finds solutions within 5% of the best for 60% of the the problems, and its ranges of outcomes are among the best for trussed arch problems, Warren truss problems, and, to a lesser extent, basket-handle arch problems (Figs. 5.5 and C.7).

Relative to COBYLA and BOBYQA, NEL-MEAD often takes longer to converge on problems where it finds particularly good solutions than on problems where it finds bad ones (5.10c), which may be a result of the simplex strategy's ability to avoid local optima. An engineer's patience when using NEL-MEAD is therefore more likely to be rewarded with a high-quality solution.

4. SUBPLEX

Subplex algorithm [Rowan, 1990]

SUBPLEX is similar to NEL-MEAD, implementing the original NEL-MEAD approach on a sequence of subspaces. Despite *a priori* expectations, SUBPLEX does not, on average, converge more quickly than NEL-MEAD. In fact, the study showed SUBPLEX to be almost always the slowest of the local algorithms to converge. Furthermore, the range of results it produces is worse than NEL-MEAD's, although it does, on occasion, find the best solution among all algorithms (Figs. 5.5 and 5.9).

SUBPLEX generally produces better solutions for mass minimization problems than for stiffness maximization problems (Fig. C.4a) and, when problems are segmented by the underlying structural type, SUBPLEX exhibits notably good performance on basket-handle arch and, to a lesser extent, suspension bridge problems. It makes relatively fewer analysis calls for trussed arch and suspension bridge problems, and finds better solutions to increasingly-constrained problems. On many of the problems where SUBPEX does not perform well, it also takes a long time to converge (Fig. 5.10d).

5. PR-AXIS

Principal Axis method [Brent, 2002]

PR-AXIS is, essentially, an adaptation of the well-known *Conjugate Gradient Descent method.* Like SUBPLEX, PR-AXIS frequently makes very many analysis calls to arrive at a relatively poor solution (Fig. 5.10e).

It exhibits a middling overall performance in terms of the quality of its design outcomes, although it is fairly often close to the best solution for mass minimization problems. It generally performs much worse on the stiffness maximization problems, however (Figs. 5.5 and 5.11). When problems are segmented by the underlying structural type, PR-AXIS is relatively strong on Warren truss problems, and notably weak on girder bridge problems.

6. DIRECT

Dividing Rectangles Method [Jones et al., 1993]

Overall, DIRECT is one of the least effective, and among the least efficient, of the algorithms (Figs. 5.5 and 5.9). It comes very close to finding the best solution for all girder bridge problems (Fig. C.7), however, and its range of performance measures on the other problems includes some occasional good solutions. Its quality on sizing problems exceeds that on geometry problems.

A challenge for an algorithm selection scheme using DIRECT would be to identify problems that correspond to the 9% of these problems where DIRECT comes within 5% of

the best answer.

7. CRS

Controlled Random Search [Kaelo and Ali, 2006]

CRS is a variant on the classic genetic algorithm. It most frequently finds the best solution and could, in this way, be considered a better algorithm than COBYLA. But its frequent strong performance is accompanied by a downside: on those problems where it is not the best, CRS is not within the top 5% of performers nearly as often as COBYLA, and its population-based approach requires many more analysis calls to produce such good results (Figs. 5.4 and 5.5). CRS is, in a sense, a high-risk choice. Using it all the time may lead to a set of designs containing the greatest number of 'optimal' solutions, but the downside associated with those instances where CRS is not the best algorithm is significant.

Interestingly, CRS performs much better, on average, on stiffness maximization problems than on mass minimization ones (Figs. 5.11 and C.3a). Among these mass minimization problems, CRS performs very badly on problems that have constraints on nodal displacement only. When stress constraints on every element are introduced, CRS's performance appears to improve. On problems with low-dimensional design spaces and computationally-inexpensive structural analyses, such as girders and basket-handle arches, CRS produces the best range of solutions (Fig. C.7).

8. ISRES

Improved Stochastic Ranking Evolution Strategy [Runarsson and Yao, 2005]

ISRES has a similar (population-based) computational strategy to CRS, and accordingly makes very many analysis calls. The average quality of its solutions is not as good as CRS's, although there are plenty of occasions when it does outperform CRS and the other algorithms (Fig. 5.9).

Like DIRECT, ISRES finds unusually good solutions for girder bridge problems; its solutions to the Warren truss and trussed arch problems are also, on average, of high quality (Fig. C.7).

5.4 Chapter summary

Using data visualizations and statistical tests, this chapter first presents overall trends in algorithm performance across problems, and then characterizes the association between important problem features and performance. The chapter's latter sections, §5.2 and §5.3, each answer one of the dissertation's principal research questions.

§5.2 confirms the assertions made in the first chapter regarding the importance of the algorithm selection problem. No single algorithm outperforms all others, performance varies significantly across types of problems and across algorithms, and there is significant benefit, in terms of solution quality and required computational cost, to choosing the right algorithm. This section also improves upon a key limitation of much of the existing literature. By showing the performance of algorithms with their default parameters on a wide range of problems, we present a clear picture of optimization's promising, but variable, potential to improve design outcomes and processes.

§5.3 examines correlations between features of the design problems and each algorithm's performance on those problems, and summarizes the major trends. This understanding of performance variation across realistic design problems is an important first step in guiding engineers' choice of algorithms, fulfilling an identified need in bringing optimization closer to practical design. The relationships between features and performance are complex and interdependent, however, motivating the next chapter's study of automated algorithm selection using sophisticated statistical techniques.

Chapter 6

Techniques for automatic selection of algorithms

Chapter 5's exploration of algorithm performance trends builds intuition and understanding, but is limited in a number ways. Although a useful step in providing guidance to engineers using optimization in design, its approach requires designers to carefully assimilate its lessons before it can beneficially impact the field; this may be an unrealistic expectation. Furthermore, the previous two chapters showed the complexity of the algorithm selection problem. Design spaces are complicated, different algorithms produce very different solutions, and the associations between problem features and algorithm performance are nonlinear, interdependent, and not easily reduced to a simple set of guidelines.

These considerations motivate this chapter's research goal: the development of computational techniques to automatically select the best algorithm for a given structural optimization problem. The chapter answers the dissertation's third, and final research question: *"Can we create a system to automatically select good optimization algorithms for design problems?"*

Chapter 2 reviewed the algorithm selection literature from optimization and from other computational fields, noting that the problem is usually approached with machine learning-based classification. The work we present is developed with those precedents in mind, but is uniquely adapted to the problem at hand. §2.2.3 identified a tendency in the literature to evaluate selection systems based only on intermediate results such as the frequency with which a system selects the best algorithm. We therefore take care to go beyond such measures and examine the impact on the design process in terms of solution quality and computational cost.

§6.1 describes two distinct formulations of the problem, and the machine learning methods

used to implement them. All the algorithm selection techniques are built using Chapter 3's data set; they draw on trends and insights from Chapter 5 and on precedents from the literature. §6.2 evaluates the selection techniques, first in terms of how often they select the best algorithm, and then in terms of their impact on design outcomes and computation times. The former is, ultimately, more important to designers in practice.

Compared to the previous chapter, the focus of the work now shifts from understanding and explaining why algorithm performance varies towards developing reliable techniques to automatically recommend algorithms, regardless of the transparency of their decision methodology. As long as their predictive power is strong, algorithm selection techniques will not be penalized for opacity.

6.1 Algorithm selection methods

Algorithm selection is formulated as both a pattern classification problem and a regression one, described separately in §6.1.1 and §6.1.2. In both approaches, the goal is to recommend the algorithm $\hat{a}_j$ that is, in some sense, best for the j^{th} optimization problem.

This section presents the two general formulations, the specifics of the statistical methods used to implement them, and some data pre-processing operations. Several methods of evaluation and a presentation of results appear in §6.2.

We first identify a few relevant terms and concepts.

Terminology and notation

Each optimization problem is characterized by a vector of *explanatory variables*, $\vec{x}$, consisting of the primary features in §3.1's experimental design—*structural type, span, height-to-span ratio, objective type,* and *nature of the design variables*—and some of the secondary features identified in the same section—*number of design variables, number of stress constraints, number of displacement constraints,* and *number of mass constraints.*

The dependent quantity being predicted is termed the *response variable.* The ultimate goal is always to predict the best algorithm for a given problem, so the response variable must be an element of the set $\mathcal{A}$ = {COBYLA, BOBYQA, NEL-MEAD, SUBPLEX, PR-AXIS, DIRECT, CRS, ISRES}. The true best algorithm for problem j is denoted a_j^*, while the predicted best algorithm is denoted $\hat{a}_j^*$.

The second of the two problem formulations uses nonlinear regression to predict an algorithm's performance on a problem, and uses these intermediate predictions to choose

the best algorithm. Under this formulation, the intermediate response variable—the predicted performance of algorithm a on problem j—is denoted $\hat{y}(a)_j$, and is compared to the true performance of algorithm a on problem j, $y(a)_j$. y can refer to either solution quality or computational cost, and subscripts distinguish the two performance measures where necessary—y_o for solution quality (objective value) and y_c for computational cost (number of analysis software calls).

The data, in which each optimization problem is represented by a vector of explanatory variables and a response variable, are split into *training* and *test* sets. The training data are used to build and tune algorithm selection techniques, and the test data are used to evaluate the techniques' predictive power.

The bias-variance tradeoff

All predictive statistical modeling of this nature should strike a balance between two potential sources of error: bias and variance. The least complex methods of prediction, often based on linear models, generally have high bias, with large differences between expected response predictions and true responses across the training data. Such models, however, have low variance, and make similar predictions for new data points across different realizations of the model. Highly complex models may closely capture the structure of the data on which they are trained, minimizing bias error, but exhibit high variance on new test data. A good statistical model balances the two sources of error.

These somewhat abstract statistical concepts hold important lessons for the fitting of predictive models. It is generally possible to closely, even perfectly, fit a model to the training data by arbitrarily increasing the model's complexity. Such low-bias models, however, *over-fit* the data, have very high variance, and do not generalize well to new

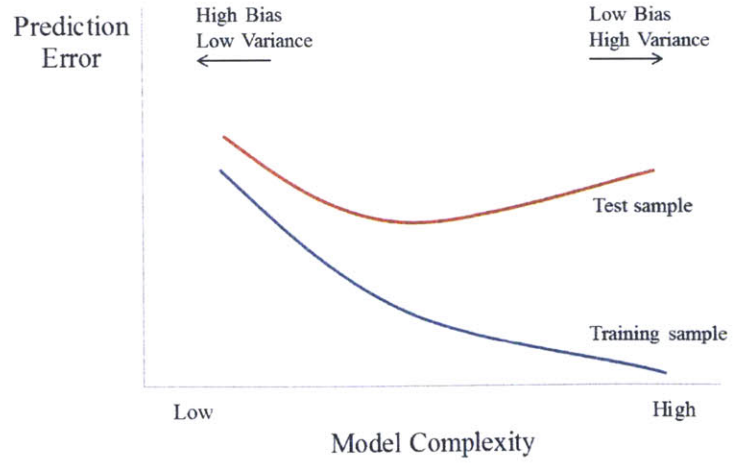


Figure 6.1 – Visualization of the bias-variance tradeoff and the concept of over-fitting (adapted from Hastie et al. [2005]).

data instances (Fig. 6.1). Strong performance on training data is no guarantee of future performance and should not be exclusively pursued or used to promote a certain method.

All the methods used in the study are tuned to minimize test set error; a method's performance on training data is noted to ensure it does not become excessive, but does not receive significant attention.

6.1.1 Classification formulation

Drawing on the experience of algorithm selection researchers in many other fields, the problem is first formulated as one of pattern classification [Hastie et al., 2005; Smith-Miles, 2008]. Pattern classification's purpose is to develop a predictive method which can assign new data observations (optimization problems) to one of a fixed set of classes.

A number of common machine learning classification algorithms, described below, are used. All of them require data points to belong to a one class only. The eight algorithms (§3.2.1, listed in the set $\mathcal{A}$ above) are defined as the possible classes, and each of the 474 optimization problems is assigned to the algorithm that performs best when solving that problem. In situations where two algorithms either produce the same solution or make the same number of analysis calls, the other performance measure is used to break the tie. There are no instances in the data of two algorithms producing the same solution while making the same number of analysis calls.

A pattern classifier's task is, therefore, to produce a mapping $f\colon \mathbb{R}^m \mapsto \mathcal{A}$, from the training data, which predicts a single class label, or algorithm, for an optimization problem (Eq. 6.1).

$$\hat{a}_j^* = f(\vec{x}_j); \qquad \hat{a} \in \mathcal{A}; \qquad \vec{x} \in \mathbb{R}^m \tag{6.1}$$

where $\hat{a}_j^*$ is the predicted best algorithm for the j^{th} optimization problem, whose characteristic features form the m-length vector of explanatory variables $\vec{x}_j$.

Supervised machine learning methods for classification

The pattern classification problem is solved with six different supervised machine learning algorithms: Binary Decision Tress, Ensemble Binary Trees, K-Nearest Neighbor Classification, Artificial Neural Networks, Support Vector Classification, and Naïve Bayes Classification. Hastie et al. [2005] provide a thorough review of their underlying theory and of relevant statistical concepts.

Unless otherwise stated, MATLAB®'s [2012] implementations of the machine learning methods are used. Their output is sensitive to certain tuning parameters and inputs; these are set manually, using a combination of literature recommendations, intuition, and trial-and-error. The goal in doing so is always to improve predictive performance on the test data; strong predictive performance on the training data usually follows. Details and parameter values are noted in each method's description below.

(a) Binary Decision Trees. We use Breiman et al.'s [1984] *CART* binary decision tree algorithm to assign labels to problems. CART proceeds by identifying the value of a single explanatory variable which binarily splits the training data into maximally-dissimilar subsets based on assigned class labels, and recursively applies the same procedure to these two subsets until the data can be split no further. Each node in a CART tree contains a single rule which splits the data in two based on the value of one explanatory variable. Terminal nodes are known as *leaves*; every optimization problem lies in one leaf only, and every leaf is defined by a unique path—a set of rules—through the tree. The resulting rules are then used to assign algorithms to new problems.

Trees are grown using the Gini-Simpson diversity index to identify splits in the explanatory variables [Caso and Angeles gil, 1988]. No lower bounds are enforced on the number of observations in each leaf, resulting in deep, complex trees. To avoid over-fitting, regions of these complex trees with comparatively poor classification power are pruned (two child nodes are collapsed into their parents) to the level that minimizes 10-fold cross-validation error on the training data [Hastie et al., 2005].

(b) Ensemble Binary Decision Trees. Ensemble methods, by aggregating the predictions of many individual classifiers, have been shown to improve classification quality, especially in the case of decision trees. Two such methods of aggregation are used here: *Boosting*, with the *AdaBoost.M2* algorithm [Freund and Schapire, 1996], and *Bootstrap Aggregating* in the form of Breiman's [2001] *Random Forests* algorithm.

The implementation of AdaBoost.M2 first generates 100 binary decision trees in a manner identical to the single tree described above, except that each leaf is forced to contain at least five training observations. This increases each tree's bias, but decreases overall variance. The algorithm weights each tree's predictions based on mis-classifications during training, starting with all weights set to 1.0 and re-weighting based on a tree's success after each iteration.

The implementation of the Random Forests algorithm, which we expect to have lower variance error and better predictive power than a single decision tree, combines the predictions of 150 binary decision trees. Each tree is trained on a set of data randomly sampled with replacement from the training set, using twelve randomly-selected explanatory variables. The trees in this aggregation are trained in the same

way as the single decision tree, with no restrictions on the number of observations in a leaf.

(c) K-Nearest Neighbor Classification. *K-Nearest Neighbor* (KNN) classification is one of the most easily-understood classification algorithms. A new observation is classified in the same way as the majority of the K points closest to it in the training set. Ties in this majority vote are broken by random selection among equally-supported candidates. The standard Euclidean measure is used to exhaustively evaluate distances between observations (with MATLAB's *Exhaustive Searcher* algorithm). To minimize the mis-classification of test data, K is set equal to 5 through trial and error.

(d) Feed-Forward Artificial Neural Networks. We use a Feed-Forward Artificial Neural Network (NN) with two hidden layers containing 20 *neurons* each. The explanatory variables are first normalized and converted to their principal component representation, later described in §6.1.3. Every neuron then applies a sigmoid transfer function to each observation in the training data, and a weighted combination of each neuron's output forms the input to the next neuron layer, in the case of the first hidden layer, or forms the output layer, in the case of the second hidden layer. The output layer has eight nodes, representing the eight possible algorithm labels; new observations are classified according to the largest value among these nodes.

The NN is trained on scaled and PCA-transformed data (detailed in §6.1.3) by adjusting each neuron's output weight and sigmoid function bias with Levenberg-Marquardt backpropagation [Hagan and Menhaj, 1994]. The backpropagation's target is set as zero mean-squared error between predicted and true class labels. We set a limit of 1000 on the number of epochs (iterations), a maximum allowable number of validation failures (on the 15% of training data set aside for validation) of 6, an initial μ factor of 0.001 (with increase and decrease factors of 0.1 and 10 depending on whether each iteration improves or worsens performance), and $\mu_{max} = 10 \times 10^{10}$.

(e) Support Vector Classification. Support Vector Machines (SVMs), which can be used for regression or classification, construct a series of maximally-separating hyperplanes between training observations. Support Vector Classifiers (SVCs) can only make predictions of a binary nature, so the *LIBSVM* library's implementation of the C-SVC *one-against-all* algorithm is used to train eight different binary classifiers, each of which predicts whether the new observations belong to a given class in $\mathcal{A}$ or not [Chang and Lin, 2011]. The class label with the strongest positive prediction among the eight classifiers is assigned to a new observation.

Before constructing hyperplanes, a radial basis kernel of the form $e^{-\gamma|\vec{u}-\vec{v}|^2}$ maps the vectors $\vec{u}$ and $\vec{v}$ away from the original variable space, increasing the likelihood of

strong separation. Trial-and-error, with the goal of minimizing cross-validated test error, suggests the setting of $\gamma = 0.125$ in the kernel function and $C = 2$ in C-SVC's cost function.

All support vector machines are trained on scaled and PCA-transformed data (§6.1.3).

(f) **Naïve Bayes classification.** Naïve Bayes classification uses Bayes's theorem to compute the posterior probability that an observation belongs to a given class. The prior probabilities are computed as the sample frequency of each class in the training set, and the likelihood is decomposed into a product of terms with a single term for each explanatory variable.

This decomposition relies on the very strong assumption that explanatory variables are statistically independent within each class, which is almost certainly not true in this case. Nonetheless, the empirically-demonstrated strong performance of Naïve Bayes in many situations with interdependent explanatory variables justify the method's inclusion here.

Given the explanatory variables' often non-normal distributions, we use separate kernel density estimates to model each one's distribution. MATLAB assigns a normal kernel to each feature automatically, and computes kernel widths for features internally.

6.1.2 Regression-based formulation

The classification approach presented in the previous section has an important limitation: by considering a single 'best' algorithm for each optimization problem, it neglects the reality that any problem can solve any algorithm. Algorithms do not simply succeed or fail at solving a problem, and the difference between the best and the second-best algorithm may be very small. Strict pattern classification discards a large amount of available information in favor of a simplistic identification of the single best algorithm.

A selection technique which takes advantage of this additional information can predict how each algorithm will perform, and then select the best one. Although this may lead to more recommendations of sub-optimal algorithms, the downside of such recommendations should, intuitively, be reduced.

To achieve this, we fit a single regression to the natural logarithm of non-normalized algorithm performance data, $\log(y)$, and include the algorithm itself as a categorical explanatory variable a. Chapter 5 showed that $\log(y)$ has approximately normal distribution for both solution quality and computational cost, which should make it easier to regress

to than y. Using this approach, the a^{th} algorithm's performance on the j^{th} problem is then predicted by (Eq. 6.2).

$$\hat{y}_j = e^{g(\vec{x}_j, a)} \tag{6.2}$$

where $g(\vec{x}_j, a)$ is the mapping produced by regressing explanatory variables on $\log(y)$ with one of the regression methods presented below. As previously noted, $\hat{y}$ is an intermediate response variable which can be used to predict the ultimate response, $\hat{a}^*$. The predicted best algorithm for problem j, $\hat{a}_j^*$, is the one with the lowest $\hat{y}_j$ (Eq. 6.3).

$$\hat{a}_j^* = \underset{a}{\operatorname{argmin}}\, \hat{y}_j(\vec{x_j}, a) \tag{6.3}$$

Alternatively, this regression-based formulation can be used to recommend the set of algorithms $\hat{a}_j^{R*}$ whose predicted performance on problem j lies within $R\%$ of the algorithm with the best predicted performance (Eq. 6.4).

$$\hat{a}_j^{R*} \in \{a \mid \hat{y}_j(\vec{x}_j, a) \leq \left(1 + \frac{R}{100}\right) \hat{y}_j(\vec{x}, \hat{a}^*)\} \tag{6.4}$$

Regression methods

(a) Feed-Forward Artificial Neural Networks. The NN used for regression is similar to the pattern classification one (§6.1.1), except that it contains only a single hidden layer of 18 neurons and an output layer with one node, since the response variable is, in this case, the single logarithm of algorithm performance rather than eight class labels.

The same version of Levenberg-Marquardt backpropagation with the same tuning parameters is used to minimize the network's mean-squared prediction error, and 15% of the training data are used to validate performance.

(b) Support Vector Regression. The LIBSVM library's implementation of Vapnik's ϵ-Support Vector Regression (ϵ-SVR) is also used to regress $\log(y)$ on the explanatory variables [Chang and Lin, 2011; Vapnik, 1999].

ϵ-SVR, in this case, uses a radial basis kernel with $\gamma = 0.125$ to transform the input points, and aims to bound a prediction of $\log(y)$ to within a tolerance, ϵ, of 0.1, with as flat a function as possible. The cost parameter, C, is set by trial and error to 2.0 in order to minimize mean-squared test error under 5-fold cross validation.

(c) Generalized Linear Model. Generalized Linear Models (GLM) extend the concept of standard linear regression by fitting a linear model to a transformation of the response variable, transformed via a *link function*. GLMs also allow for assumed probability distributions other than normal.

MATLAB's *Backward Feature Selection* algorithm is used to progressively remove terms from a full quadratic model, assuming a normal distribution for $\log(y)$.

(d) Regression Tree. Regression Trees operate similarly to the classification trees used in the previous formulation. They recursively split the design space into maximally-dissimilar regions, except dissimilarity is now measured in terms of the continuous response variable rather than class labels. The implementation used here fits a constant model to $\log(y)$ at each node.

6.1.3 Data pre-processing

Many of the machine learning algorithms used in both of these formulations do significantly better when trained on pre-processed data. Three pre-processing operations are used: dummy variable encoding of categorical variables, variable scaling, and explanatory variable selection.

Dummy-variable encoding is used in all cases; the categorical nature of several explanatory variables (e.g., structural type, nature of design variables, and objective type, all of which are inherently discrete), requires such encoding before the methods can be correctly applied. The cases where variable scaling and feature selection are used, already detailed in §6.1.1 and §6.1.2, are repeated in their descriptions below.

Dummy variable encoding

Three of the explanatory variables—*structural type, objective type,* and *nature of the design variables*—are categorical (qualitative), taking on values in a fixed set. For example, *structural type*, a six-level categorical variable, always takes a value in the set {Trussed Arch, Basket-Handle Arch, Cable-Stayed Bridge, Girder, Suspension Bridge, Warren Truss}.

Dummy variables are the most commonly-used encoding strategy for variables of this nature [Hastie et al., 2005]. They represent a K-level categorical variable by a vector of $(K-1)$ binary variables. Setting the i^{th} variable to 1 means the categorical variable takes on the i^{th} possible discrete value, and setting all variables to 0 means the categorical variable takes on the K^{th} value. No more than one dummy variable is set to 1 at a time.

Applying dummy variable encoding to a K-level categorical variable therefore increases the number of explanatory variables in the problem by $(K-2)$.

The encoding is applied to all categorical variables for all algorithm selection techniques, leading to a vector of explanatory variables $\vec{x}$ containing fourteen elements, the first eight of which are binary (Eq. 6.5).

$$\begin{aligned}\vec{x} = (&\text{Type=TrussedArch, Type=BasketHandleArch, Type=CableStayed, Type=Girder}\\ &\text{Type=Suspension, DesignVariables=Geometric, Objective=Stiffness, Span,}\\ &\text{Height/Span, \#design variables, \#displacement constraints, \#stress constraints,}\\ &\text{\#mass constraints)}\end{aligned} \tag{6.5}$$

Scaling of variables

The explanatory variables have very different numerical ranges. Observed values of bridge span, for example, range from 50 m to 800 m, while the number of design variables never exceeds 14. Such discrepancies can cause numerically-larger variables to dominate when using certain methods. To avoid this, each explanatory variable, x, can be normalized to have zero mean and unit variance according to Eq. 6.6.

$$\bar{x}_{ik} = \frac{x_{ik} - \mu_k}{\sigma_k} \qquad i = 1 \ldots N; \qquad k = 1 \ldots e \tag{6.6}$$

where μ_k and σ_k are the mean and standard deviation of the k^{th} explanatory variable calculated across all of N problems.

When using a system trained on scaled data to make predictions, the problem's explanatory variables must first be scaled by the same μ_k and σ_k used to scale the training data. This is an important consideration in properly testing a selection technique's performance by cross-validation, described later in §6.2.

The data are normalized when using Neural Networks or Support Vector Machines in either of the two problem formulations, and when using Generalized Linear Regression for the regression-based formulation.

Explanatory variable selection

Variable selection is the identification of the subset of explanatory variables which best predict the response. This is achieved by successively removing low-information or confusing explanatory variables, or by converting variables to a reduced form.

The latter approach, in the form of Principal Component Analysis (PCA), is adopted before training Neural Networks and Support Vector Machines. PCA, although technically a transformation of the explanatory variables, nonetheless satisfies the same purpose as straightforward variable selection. It converts the original variables into a set of linearly-uncorrelated ones, ordered by their variance. This usually reduces the number of variables, and it is not uncommon to then remove low-variance ones after transformation [Jolliffe, 2002]. Even when the number of variables remains constant, PCA is effective in transforming low-information explanatory variables to more useful ones.

As with normalization of variables, new problems' explanatory variables must be transformed by the same PCA operations used on the training data before any predictions are made. When data are pre-processed using both variable scaling and PCA, normalization is always performed first.

MATLAB's processpca function is used to apply PCA, removing variables whose variance is less than 0.01 after transformation.

The only other instance of programmatic feature selection is when fitting generalized linear models (GLMs) in the regression-based formulation. This approach, using MATLAB's *Backward Feature Selection* algorithm, was noted in the GLM section above (§6.1.2).

6.2 Evaluation and results

We first evaluate the results with the standard binary measure, where a selected algorithm is deemed either a success or a failure depending on whether it is the true best. This is the method most commonly used in classification theory, and is a useful intermediate evaluation of the developed techniques. A custom solution gain measure is next used to evaluate results, quantifying the overall decrease in algorithm performance compared to an idealized perfect selector, which always chooses the best algorithm (§6.2.1).

§6.2.2 then directly quantifies the impact of the selection techniques on design outcomes, evaluating the potential material and cost savings compared to traditional optimization. In all cases, the developed selection techniques are compared to several manual algorithm strategies which a designer might use.

6.2.1 Success rates and solution gain

This section details the evaluation of binary success rates and of overall solution gain. For each technique, we evaluate how well it generalizes to independent data using 10-fold cross-validation [Hastie et al., 2005]. One round of K-fold cross-validation uses $(100-K)\%$ of the data to train the selection technique, and tests how often this trained technique selects the best algorithm for the remaining $K\%$ of the data. This is repeated K times with different partitions, and the test data success rates for each are averaged.

For completeness, each selection technique's training success rate—the rate at which it selects the best algorithm for problems on which it is trained—is also reported, although this measure is less important than cross-validated test success rate. As mentioned in §6.1, high test success rates are easily achieved by arbitrarily increasing model complexity.

Binary success rates are widely reported in statistical research of this nature, and are the dominant means of evaluation in the algorithm selection literature (Chapter 2). One of the major limitations of a black-and-white measure of success, however, is the inability to distinguish between algorithms that are narrowly outperformed and those whose performance lags far behind the best. An algorithm selection system that infrequently selects the best algorithm but is less likely to recommend a badly-performing one may be preferred to a system with greater downside.

To address this limitation, we first express the performance of the chosen algorithm, $y_j(\hat{a}_j)$, as a proportion of the true best algorithm's performance on the j^{th} problem, $y_j(a_j^*)$ (Eq. 6.7)[1].

$$C(\hat{a}_j) = \frac{y_j(\hat{a}_j) - y_j(a_j^*)}{y_j(a_j^*)} \tag{6.7}$$

Higher values of $C(\hat{a}_j)$ represent worse performance, in the form of worse solution quality (less stiff or heavier designs) or more analysis software calls. These values are used in calculating a cumulative gain measure $\mathcal{G}$ over the N_{test} test problems Eq. 6.8.

$$\mathcal{G} = \sum_{j=1}^{N_{test}} \left(1 - 2^{-1.5} C(\hat{a}_j)^{1.5}\right) \tag{6.8}$$

Because of the power term $C(\hat{a}_j)^{1.5}$ in the gain function, algorithms whose performance is close to best contribute similarly to overall gain, and the minimum possible gain gradually

[1] As in the previous chapter, stiffness objective values (Eq. 3.2) are converted to maximum displacement to enable meaningful calculation of magnitude relative to the best design.

moves towards a lower bound of zero. As in the previous chapter (§5.2), any time an algorithm's performance is more than 200% worse than the best algorithm, the solution is discarded. This is equivalent to placing an upper bound of 2.0 on the proportional performance $C(\hat{a}_j)$, limiting the incurred penalty for selecting a bad algorithm. Some algorithms have solution qualities and computational costs which are orders of magnitude worse than the best algorithm; removing outliers prevents their selection from dominating the gain measure. Besides, engineers can generally predict the best solution quality within an integer multiple or so (using material use plots such as Fig. 4.1), so there is little point in continuing to penalize a solution that uses over 200% more material than the best answer and will almost certainly be discarded.

The gain function has little direct physical significance, and should be primarily used for relative evaluation of algorithm selection techniques. A perfect technique, which always finds the best-performing algorithm, has $\mathcal{G} = N_{test}$. A technique whose selected algorithm is always 10% worse than the best algorithm over N_{test} test problems has $\mathcal{G} = N_{test} \times (1 - 2^{-1.5} \times 1.1^{1.5}) = 0.592 N_{test}$, and the minimum possible gain over N_{test} problems is zero.

Baseline for evaluation: manual algorithm selection strategies

Understanding the real impact of this work requires a quantification of the current state of algorithm selection in structural design. We define two 'manual' selection strategies, both representative of how a designer might choose an algorithm to solve a given problem, and use their success rates and gain measures as baselines against which to compare the developed selection techniques.

These two strategies are: constant use of a single algorithm, and choosing algorithms at random based on a uniform probability distribution. For the first strategy, we use both CRS (the most frequently top-ranked algorithm) and COBYLA (the most consistent algorithm) for comparison when attempting to select the algorithm with the best solution quality, and BOBYQA when selecting the computationally-lightest algorithm. This could be argued as an unfairly-difficult baseline, since it makes use of posterior knowledge from Chapter 5's explorations, which a designer would not have conducted in advance. Convincingly outperforming this strategy, therefore, should be ample evidence of success.

Results: success rates and gain measures

Table 6.1 shows each selection technique's binary success rate (100% means the best-performing algorithm is always selected) and gain measure, $\mathcal{G}$ (Eq. 6.8).

The left-hand data columns contain test results from 10-fold cross validation. Test suc-

cess rates are averaged over the ten cross-validation sets, and test $\mathcal{G}$-values are summed over the same sets. Each test gain measure is therefore the sum of $N_{test} = 474$ components, each of which contributes no more than 1.0 to overall gain, for an upper bound of $\mathcal{G}_{max} = 474$. Training results, shown in the right-hand column, reflect how well the techniques fit the data on which they were trained, and are therefore less useful in evaluating predictive power. They are included in Table 6.1 for completeness, but the remainder of this discussion refers to test results only.

As expected, 'manually' selecting algorithms at random delivers the best-quality algorithm in $\frac{1}{8} = 12.5\%$ of cases. (There are eight algorithms in the set of possible algorithms, $\mathcal{A}$.) Choosing the algorithm most frequently ranked best on all the problems—CRS—gives the best solution in 40.0% of cases, and choosing the most consistent performer—COBYLA—gives the best solution in 36.0% of cases. These values agree with the counts shown in Fig. 5.4.

Table 6.1 – When seeking the algorithm that generates the best solution, the best developed technique is successful more than twice as often as manual selection strategies on independent test data, and has a substantially higher cumulative gain value $\mathcal{G}$.

Algorithm selection technique	Cross-validated test data		Training data	
	Binary success (%)	Gain, $\mathcal{G}$ (Eq. 6.8)	Binary success (%)	Gain, $\mathcal{G}$ (Eq. 6.8)
Manual selection				
Manual - random choice	12.5	275.2	12.5	292.4
Manual - always COBYLA	36.0	412.3	36.0	98.5
Manual - always CRS	40.0	322.4	40.0	277.7
Pattern Classification formulation				
(a) Binary decision tree	73.2	434.3	79.3	442.6
(b) Boosted decision tree	71.1	435.0	99.8	461.8
(b) Random Forest	74.7	436.9	100.0	461.8
(c) K-nearest neighbor	70.6	433.1	77.5	440.7
(d) Artificial neural network	64.6	420.0	44.9	415.7
(e) Support vector classification	67.6	429.9	82.9	444.1
(f) Naïve Bayes	61.4	422.8	62.9	429.4
Regression-based formulation				
(a) Artificial neural network	22.7	160.9	32.0	167.8
(b) Support vector regression	54.2	332.9	58.9	384.2
(c) Regression Tree	63.9	413.1	84.2	420.2
(d) Generalized linear model	28.5	160.4	28.3	161.4
Best observed value	**74.7**	**436.9**	**100.0**	**461.8**
Improvement over manual selection	34.7	24.6	60.0	169.4

Choosing COBYLA leads to dramatically higher solution gain $\mathcal{G}$ across the test data, meaning that COBYLA's solutions are, on average, closer to the best solution than CRS's or than the solutions generated by randomly-chosen algorithms. These values are the baselines to which the selection techniques' results should be compared.

Most pattern classification techniques comfortably outperform the manual selection strategies; some choose the best algorithm almost twice as frequently. The explicit goal of these techniques is to maximize this success rate, and such strong performance is welcome evidence of an exploitable mathematical relationship between the features of an optimization problem and the identity of its best algorithm.

Of greater importance, of course, is the overall quality of the solutions, measure by $\mathcal{G}$. All the pattern classification techniques have higher solution gains than the best manual strategy[2], although some outperform it by relatively narrow margins. Naïve Bayes classification, for example, selects the best algorithm much more often than the best manual strategies, yet its overall solution gain is only slightly better. In the cases where it does not select the best algorithm, therefore, Naïve Bayes must be choosing algorithms which produce quite poor solutions.

On the whole, however, the results indicate that algorithm selection using pattern classification leads to better-quality solutions than manual strategies that represent the state of optimization in practice today. These results are also a useful indicator of the disconnect between frequently choosing the best algorithm and frequently producing good designs. A selection technique's usefulness is in part governed by the performance of its sub-optimal choices.

Most of the regression-based approaches are much less impressive than the pattern classification ones in terms of binary success rate and solution gain. Their immediate goal, of course, is to predict performance, so the lower binary success rates alone should not be too alarming. Indeed, despite predicting the best algorithm only 63.9% of the time (much less frequently than the pattern classification methods), the best of the regression-based methods—the Regression Tree—still has substantially better solution gain $\mathcal{G}$ than manual strategies. The downside-minimizing approach of the regression-based formulation means that, on the frequent occasions when the Regression Tree does not select the best algorithm, it selects one whose performance is not too far behind.

The other regression-based methods, however, perform much worse—in terms of the frequency with which they select the best algorithm and the quality of the solutions those algorithms produce—than simply using COBYLA or CRS for all problems. Despite this poor performance, the highly promising results of the Regression Tree show the value of approaching the algorithm problem with the proposed regression-based formulation.

[2]Each problem's contribution to $\mathcal{G}$ lies in the range $[0, 1]$, so we can safely assume these measures reflect overall solution quality and are not dominated by a single problem.

All of the preceding discussion dealt only with selecting the algorithm that produces the best solution, neglecting selection of the computationally-lightest algorithm. We consider the speed of an algorithm to be less important than the quality of its output, and do not examine these results in the same level of detail. They are instead tabulated in Appendix D (Table D.1), and show comparable binary success rates to the design quality results presented here. Table D.1 also shows how often a technique chooses an algorithm whose required number of analysis calls is within 10% of the fastest algorithm.

6.2.2 Impact on bridge design quality

The gain measure $\mathcal{G}$ used to evaluate selection techniques in the previous section gives a sense of the improvement in solution quality that results from replacing current manual selection strategies with the developed selection techniques. This measure, however, is somewhat arbitrarily defined and does not sufficiently quantify the design improvements that result from using automated algorithm selection. This shortcoming occurs often in the algorithm selection literature; researchers generally quantify how often their system selects the best algorithm, but do not translate this to a tangible improvement in the design process.

Here, we compare the solutions generated by automatically-selected algorithms to those generated by 'manually'-selected algorithms. To enhance the comparison's relevance to engineers, we now address the mass minimization problems only, to determine how much material could be saved by using the developed techniques.

The outcomes for one of the best automated selection techniques—Boosted Binary Decision Trees—are compared to those for the best manual selection strategy of using COBYLA every time. The 474 problems are randomly split into a 374-problem training set and a 100-problem test set. The boosted decision tree classifier is trained on the training set and used to select the best algorithm for the 100 test problems. 35 of these problems have mass as their objective function; Fig. 6.2 shows the reduction in material used compared to the manually-generated starting points for both the predicted algorithms and for COBYLA alone across the 35 problems.

In twelve of the 35 cases, the classifier chooses an algorithm that produces a design of the exact same mass as COBYLA. (The classifier is, of course, free to choose COBYLA.) In five cases, the classifier's chosen algorithm produces a worse solution than COBYLA, using over twice as much material in one instance. (This instance, however, is an outlier; on the four other occasions where COBYLA outperforms the classifier's chosen algorithm, COBYLA saves between 2% and 30% more material.) For the remaining eighteen problems, the classifier chooses an algorithm that uses substantially less material than COBYLA, and an average material reduction of 9.4% is observed (Fig. 6.2).

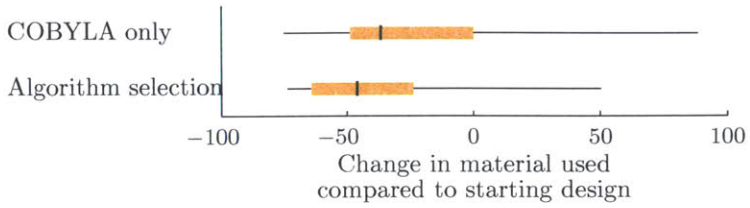


Figure 6.2 – The amount of material used when algorithms are automatically chosen decreases, on average, by 9.4%.

Using the developed classifier to select algorithms instead of the best-performing manual strategy results in lighter solutions in a majority of cases, with average material savings of 9.4%.

6.3 Discussion

The automatic algorithm selection techniques presented in this chapter achieve the stated goal of outperforming manual algorithm selection, generating better solutions and reducing computation times. This directly addresses the need, motivated in the dissertation's first two chapters, to adapt optimization to the reality of practice by allowing non-experts to reliably choose an appropriate algorithm for a given problem.

Even when evaluated against manual selection strategies based on posterior knowledge of the dataset, the most promising of these techniques nearly doubles the rate at which the absolute best algorithm was selected and substantially reduces a custom cumulative gain measure across a range of optimization problems.

Considering only the mass minimization problems, one of the developed techniques reduces mass in over half the cases, and causes no increase in material use in a further 35% of cases. In the remaining 15% of mass minimization problems, the selection system results in a worse solution, but the average additional material reduction across the problems is still 9.4%. This may seem small compared to some of the large inter-algorithm variations observed in the dissertation. Consider, however, a 1000 m bridge design project, where superstructure material costs could total $10M. A 9.4% material reduction in this case translates to a saving of $940,000M, achieved through a process which frees engineers from tedious, low-level design iterations.

Fully-automated algorithm selection based on machine learning, in contrast to the pedagogical approach of Chapter 5, is largely opaque. The machine learning methods are, from the perspective of a typical system user, 'black boxes' which specify an algorithm

without providing any domain-specific logic for doing so. Although this does not diminish the demonstrated results, it does warrant a consideration of the context in which an automated selection system would be used. A designer encountering a poorly-performing algorithm is likely to be far more frustrated if the algorithm was recommended by a predictive system than if she chose it herself.

The results show that this is an entirely possible occurrence; even the best among the selection techniques occasionally chooses an algorithm that produces a very poor solution. The techniques have been shown here to objectively outperform manual algorithm selection but, in promoting them, we should remember that their reception among design engineers will be based, in large part, on subjective assessments. This motivates future investigation of methods to reduce their downside.

Chapter 7

Conclusion

7.1 Review

The dissertation's first chapters, having shown the potential of optimization to improve design processes and outcomes, identify the choice of algorithm as an important part of a structural optimization process. Chapter 2's literature review shows the availability of many algorithms for a wide variety of problems, and the dramatic effect of algorithm selection on solution quality and computation time is demonstrated in an empirical study. Despite its importance, the problem of appropriately selecting algorithms has received little attention, reflecting a broader failure to adapt academically-developed optimization methods to the realities of practical design. This situation motivates the dissertation's research goals, expressed in three clear questions:

- How do different optimization algorithms perform on a representative set of realistic structural design problems?
- Do correlations exist between problem features and algorithm performance?
- Can we create a system to automatically select good optimization algorithms for design problems?

Chapter 3 then describes the development of a computational method to specify, solve, and record hundreds of structural optimization problems with a representative set of algorithms; the scale of this task leads to the development of several novel computational elements.

In answering the first two research questions, the generated data are explored in Chapters 4 and 5. The first of these shows the optimization results to be rational solutions to

real-world problems, enhancing the practical applicability of the remainder of the work. The second exposes high-level trends in algorithm performance, building comprehension of the problem. Algorithms display a consistent pattern of design improvement, although solution quality and computational cost depend strongly on the choice of algorithm and on the problem at hand. Chapter 5 then identifies major correlations between problem features and both solution quality and computational cost. The results section concludes by recommending the problem categories on which particular types of algorithm should be used.

In response to the third question, and to the perceived limitations of this recommendation-based approach, the focus shifts in Chapter 6 from understanding performance trends to developing black-box techniques to select algorithms. This allows sophisticated methods to handle the demonstrated complexity of algorithm selection, and provides a practical solution for immediate use in optimization-driven design. The developed techniques convincingly outperform the current manual approaches to the problem, selecting the correct algorithm much more frequently, improving structural designs, and reducing computation times.

This final chapter specifies the key findings of the presented research and enumerates the major contributions to the field (§7.2). It then progresses to discuss the work's general implications, noting its major advantages and limitations (§7.3). Finally, §7.4 outlines future research that should be undertaken, focusing first on direct extensions of this work, and then extending in scope to describe pressing challenges to widespread adoption of optimization and the path towards next-generation software for structural design.

7.2 Key findings and contributions

7.2.1 Findings

Algorithm performance exploration

Chapter 5's approach to algorithm selection is, in a sense, an educational one. To help address the problem, it explores the major trends in performance, building intuition and expertise about which algorithm should be used for a given class of problems. The chapter identified several discernible trends, reviewed in its concluding paragraphs (§5.3.6).

No single algorithm is best across all problems, even when the criteria for success is relaxed to performing within 5% of the best algorithm. There is, in short, no 'silver bullet' which works best in all cases. Algorithms generally improve on the starting designs, although the extent to which each one does so depends on the problem being solved.

Those that frequently produce the best solution sometimes produce the worst, and vice-versa. Interpreted alongside the designs examined in Chapter 4, whose solution quality can vary by orders of magnitude, this further underscores the need to choose algorithms appropriately. In contrast to the general optimization literature, our results justify a more cautious endorsement of the use of optimization in structural design.

The exploration of performance exposes several statistically significant and useful correlations between algorithms' performance and problem characteristics. Local algorithms, for instance, achieve relatively better solutions than global ones as the number of design variables increases, although the computational cost of achieving these results becomes higher. The presence of geometric design variables does not dramatically affect relative algorithm performance, unlike the objective function type (mass or stiffness), which influences solution quality and computational cost.

Structural type is also strongly associated with solution quality and computation time; each algorithm has at least one category of structure on which it performs dramatically worse than the others. Most algorithms take relatively longer to converge as the number of constraints increases; this is probably due to the complexity of the increasingly-constrained design spaces overwhelming their decreasing size. The degree of constraint, however, does not correlate significantly with solution quality.

The studied range of problems—mass and stiffness optimization of six different bridge superstructures—is a subset of those encountered in structural engineering, and the main findings strictly apply to this subset only. Wolpert and Macready's No Free Lunch theorems for optimization, however, tell us that the strong performance of, say, COBYLA and CRS on these problems will be balanced by relatively poor performance on some other set of problems. There is every reason to believe that further investigations across different types of structure will show different performance variations, supporting the conclusion that no singe algorithm is always best.

Algorithm selection techniques

Chapter 6 successfully demonstrates automated algorithm selection for structural optimization, achieving one of the dissertation's key research goals. The chapter, after formulating and implementing two approaches to the problem, shows that it is possible to significantly outperform the best strategies for manual selection of algorithms. The developed techniques nearly double the frequency with which the best algorithm is selected, dramatically improve a cumulative solution gain measure across test problems, and reduce average material use by 9.4% on a set of mass-minimization problems. The system is also capable of selecting the fastest algorithm, reducing computation times.

These results demonstrate the impact of the developed algorithm selection system on

the practice of design. The system directly addresses a key barrier to optimization's use, providing an off-the-shelf solution to the algorithm selection problem that brings engineering a step closer to improved, optimization-driven design.

The nature of structural design spaces

Chapter 4 examines the 3792 designs that result from solving 474 problems with eight optimization algorithms. The amount of material used by the best solutions is in line with existing structures, and most representative solutions chosen from clustered subsets are rational. This verification of the results shows that the dissertation's results are based on realistic problems, and also gives insights into the nature of the studied design spaces.

Many of these design spaces have multiple local optima (solutions which cannot be improved by very small design perturbations in any direction). Such non-convexity is to be expected, given the nonlinear interactions between design variables and the resulting physical behavior. Of greater interest is the difference in quality among these local optima. In some cases, design spaces have very different locally-optimal solutions which nonetheless use similar amounts of material. In other cases, however, two locally-optimal designs may use very different amounts of material.

Local optimization methods, in the absence of certain modifications, do not distinguish between local optima, converging on the first one they find. The results show that the potential negative consequences of such convergence vary by problem. Although not the primary purpose of the dissertation, this provides a useful cautionary note regarding the use of local algorithms for structural optimization.

7.2.2 Contributions

The dissertation makes a number of novel contributions to the fields of structural optimization and algorithm selection:

Trends in algorithm performance. The conclusions of Chapter 5 regarding overall performance trends and correlations between problem features and performance contribute important knowledge to the field of structural optimization. They impart several high-level messages and identify a set of important predictors of algorithm performance, reviewed in the previous section and supported in detail in the chapter itself.

These conclusions, as well as contributing to a clearer understanding of optimization's potential to impact design, allow engineers to make more informed decisions about which types of algorithms to use for a given problem.

Automated selection techniques. The developed algorithm selection techniques are the technical core of an off-the-shelf system for automatically recommending algorithms for the studied class of problems. They can be used to reliably select an appropriate algorithm, leading to improved solution quality and reduced computation times. This is a tangible step towards addressing the limitations of optimization outlined in the dissertation's introduction.

Chapter 6, in evaluating these selection techniques, explicitly focused on their design impact rather than on intermediate statistical results alone, improving on a limitation found in much of the algorithm selection literature.

Computational infrastructure. The computational infrastructure presented in Chapter 3—consisting of an optimization library, a structural analysis engine, a database system, and thousands of lines of code integrating them in the solution of thousands of algorithm problem pairings—contains several elements which are unique to the field.

One of these is the relational model for storing and retrieving optimization data, shown in Fig. 3.10. This is a unique, extensible strategy for representing structural optimization problems and algorithms, for storing results and other relevant information on algorithm performance, and for facilitating post-processing of very large data sets. As well as providing a model for future large-scale optimization research in this domain and others (the domain-specific aspects of the data model are localized and replaceable), the model could form the backbone of a practical structural optimization system which records performance data to improve future algorithm predictions and knowledge discovery.

The software wrappers around the *NLOpt* optimization library and the *GSA* structural analysis package are published online as open-source repositories, with URL addresses specified in the text. These give researchers access to a set of analysis and optimization capabilities sufficient to address many of the future research questions in structural optimization, and are therefore useful contributions to the field.

7.3 Discussion

The second half of the dissertation, which used the generated data to examine algorithm performance trends and develop automatic selection techniques, contains two very different approaches to the algorithm selection problem. In Chapters 4 and 5, the goal is to develop understanding and intuition by exploring results, whereas Chapter 6 aims to develop black-box selection systems, with little attention paid to transparency. Both contribute to the same goal of enabling engineers to select suitable algorithms, but their different philosophies have certain strengths and weaknesses.

The former, explanatory approach develops understanding of the problem and of related issues, but requires engineers to carefully read and assimilate its lessons before it can be of use. The latter, opaque approach reliably deals with complexities of the problem that cannot be reduced to a simple set of guidelines, and is shown in Chapter 6 to improve outcomes and reduce computation times. Its lack of transparency and manual input, however, may disproportionately frustrate designers on those occasions where it performs poorly.

Ultimately, both are useful. Explaining the major trends at play is an important part of educating engineers about what works in optimization, and of making the case to researchers that the variable performance of algorithms must be accounted for in adapting their work to practice. To actively address the problem and reduce the identified limitations of optimization research, the development of automated selection techniques is a more pragmatic short-term goal.

To place these findings and contributions in context, we next address some of the work's limitations, evaluating their significance and explaining why they arise.

Limitations

When discussing this work with optimization researchers, a commonly-encountered concern is the failure to repeatedly tune an algorithm's parameters (e.g., initial step size, trust region, or population size) to each individual problem, as is typically done in the optimization literature.

The first reason for not doing so is, admittedly, the infeasibility of repeatedly solving all 3792 algorithm-problem pairings while manually adjusting tuning parameters. Some problems take hours to run, and such an exhaustive exercise would have reduced the scope of the work dramatically. The most important reason for not adjusting parameters, however, lies in the reality of how optimization algorithms are used in practice. Design engineers, lacking the expertise of optimization researchers, are unlikely to repeatedly run an algorithm while modifying its parameters. In practice, algorithms are used as off-the-shelf solvers with their default settings. We use recommended settings available in the literature for all problems, which provides a more realistic representation of the real-world problem being addressed. An interesting future extension of this work, described below, is the inclusion of tuning parameters as outputs to be determined by a selection system.

Another limitation is the choice of a single starting point for each problem (Chapter 3). Some algorithms might perform better than observed here if they had started searching from many different initial designs. Again, this is partly an issue of feasibility. The computational cost of generating the data set used in this work is substantial, and using enough start points to justifiably explore their effect would have increased that already-

burdensome cost by at least an order of magnitude.

For each problem, the start point is manually generated based on rules of thumb and the author's structural design experience. Each start point is, therefore, a reasonable first attempt at solving the design problem, and is likely similar to the start point that most engineers would send to an optimization algorithm. Another factor mitigating this limitation is the averaging effect over hundreds of problems. An algorithm that encounters a start point which presents it with difficulty on one problem is likely to encounter a relatively favorable start point on another, and algorithms that perform consistently poorly here could perform as poorly with multiple restarts.

One last important limitation is that all the characteristic features identifying optimization problems apart from the number of design variables and the degree of constraint—lie in the physical engineering domain. In reality, mathematical characteristics such as the complexity of the objective function or the steepness of its gradients are more likely to govern algorithm performance and may be better explanatory variables for the selection problem.

Mathematical features, although likely to work well as performance predictors, are generally costly to compute for simulation-based optimization of this nature. Physical features such as span or structural type are instead used in the hope that they serve as proxies for the true mathematical behavior. The advantage to using physical features, of course, is that they are easily computed or easily assigned by a human operator, greatly facilitating their use as explanatory variables in a practical algorithm selection system.

7.4 Future work

We conclude by motivating future research tasks, ordered by increasing generality from direct extensions of this work to next-generation, optimization-driven, creative design tools.

7.4.1 Algorithm selection for structural optimization

This work addresses only a subset of the optimization problems encountered in structural engineering, and its impact is accordingly limited. To broaden this impact, there is a need to explore other structural types where optimization can be of benefit, such as long-span roofs, shell and spatial structures, and tall buildings. The subtleties of each of these sub-domains will likely require modifications to the techniques presented here. The effect of different objective functions (such as cost and alternative measures of stiffness) and

of additional constraints (especially on local and global buckling of structures and on dynamic frequencies) on algorithm performance should also be explored.

The work could also be extended in response to any of the limitations described in §7.3. Problems could be re-run with multiple starting points to ensure algorithms are judged fairly, the effect of algorithm tuning parameters could be accounted for, and the design space could be characterized mathematically as well as physically.

Although the set of chosen algorithms is broadly representative of those available, there are others that could be considered. Notably, none of the presented algorithms calculate gradients of the objective and constraint functions. Furthermore, algorithms could be characterized by their features (for example, whether they use a population of solutions, whether they approximate the design space, or whether they use gradients), allowing for generalization in the algorithm space as well as in the problem space.

Finally, we believe the regression-based formulation of the algorithm selection problem deserves further attention. As the first study of its kind to use the approach in this domain, §6.1.2 is a useful proof of concept, but there are many alternatives and potential refinements which may improve the results. The formulation's overall approach of predicting performance instead of directly choosing the 'best' algorithm seems inherently sensible and, when fully developed, likely to minimize the downside of an automated selection system in those instances where it does not select the best algorithm.

7.4.2 Adapting optimization to reality

Chapter 1 identified a set of factors limiting optimization's widespread application (§1.1.2). The dissertation addressed one of them, but, for optimization to truly benefit the design industry, future research should tackle several remaining limitations.

Recent developments in structural optimization have extended methods towards the real complexities of a structural engineering system. Multi-physics implementations of multiple-objective problems, sophisticated structural modeling and analysis, and the development of realistic cost and feasibility models have contributed greatly to the field's practicality, but still have some way to go before sufficiently representing the richness of a structural system. With the increasingly ubiquitous concentration of information in Building Information Modeling (BIM) representations, digital representations are becoming more complex. This presents challenges for human designers, whose cognitive limits can be quickly overwhelmed by multi-layered, extensive models, but creates opportunities for optimization-driven design to leverage the newly-available information.

By continuing to pursue full design automation, however, optimization will almost certainly encounter a problem faced in the past: some issues, especially those of aesthetics

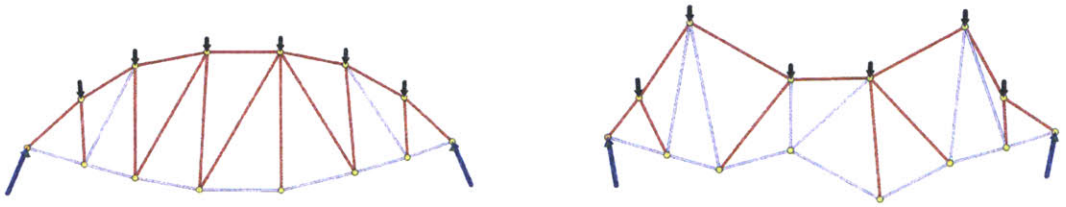
and practicality, will, for the foreseeable future, be better handled with human input. The next section briefly revisits the author's previous work on design software that combines optimization with designers' input, and shows how robust solutions to the algorithm selection problem and others like it are essential for this approach to scale to realistic and varied design problems.

7.4.3 Future design software

The author's previous work on structural optimization, and other interactive approaches, show the benefits that accrue from using optimization in a flexible, interactive way [Von Buelow, 2008; Clune et al., 2012, 2011].

Fig. 7.1a shows the result of using one of these tools to solve the mass minimization of a two-dimensional truss. Users can graphically modify the problem statement and can easily modify the resulting solution in an environment that provides real-time feedback on structural behavior and performance. Optimization reduces material usage by over 25% in this case, but is easily overriden by the designer's commands [Clune et al., 2012]. This approach follows the *Cockpit Metaphor*, which draws an analogy between autopilot systems and optimization algorithms, and is suggestive of a future for optimization-driven, but ultimately user-controlled, design [Colgan et al., 1995].

A change in the problem formulation, however (in this case, restraining the elements' end releases, which changes the structural type from a truss to a frame) can cause the same optimization algorithm to converge on a highly irrational solution (Fig. 7.1b). Indeed, experiences of this nature were an early source of motivation for working on the algorithm selection problem. Repeated experimentation with this prototype design tool lead to similar outcomes. It would often behave very well, enabling impressive demonstrations, but would frequently, and unpredictably, fail to perform well on certain problems. This



(a) (b)

Figure 7.1 – Interactive, optimization-driven software previously developed by the author often leads to rational solutions (e.g., **[a]**); slight modifications to the optimization problem, however, can lead to irrational outcomes (e.g., **[b]**) [Clune et al., 2012].

is representative of how algorithms perform on much more complex problems. Good results can be achieved, but generally require tuning of a few different algorithms using a combination of experience and luck.

Flexible and interactive use of optimization embedded in real-time simulation environments is highly desirable, and would directly address some of optimization's most fundamental limitations. In the absence of further research such as that presented in this dissertation, however, significant obstacles exist to developing such software.

We believe optimization can play a major role in design, but not before the field addresses issues that have received little attention to date. There almost certainly exist other barriers to adapting optimization to the realities of design, but addressing those outlined here would bring structural optimization much closer to making the positive impact of which it is capable.

Bibliography

Adeli, H. (1994). *Advances in design optimization.* London: Chapman & Hall. 28

American Institute of Steel Construction (2011). *Steel Construction Manual: Fourteenth Edition.* American Institute of Steel Construction, Chicago, IL. 18, 46, 47, 163, 173

Amestoy, P. R., T. A. Davis, and I. S. Duff (2004, September). Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software 30*(3), 381–388. 69

Arnold, D. C., S. Blackford, J. Dongarra, V. Eijkhout, and T. Xu (2000). Seamless access to adaptive solver algorithms. In *SGI Users Conference*, pp. 23–30. 41

Arora, J. (2011). *Introduction to optimum design* (3rd ed.). Waltham, MA: Academic Press. 28

Awad, Z. K., T. Aravinthan, Y. Zhuge, and F. Gonzalez (2012). A review of optimization techniques used in the design of fibre composite structures for civil engineering applications. *Materials & Design 33*, 534–544. 37

Balling, R. J., R. R. Briggs, and K. Gillman (2006). Multiple optimum size/shape/topology designs for skeletal structures using a genetic algorithm. *Journal of Structural Engineering 132*(7), 1158–1165. 31

Banichuk, N. V. (1983). *Problems and methods of optimal structural design*, Volume 26. New York, NY: Springer-Verlag. 28

Barthelemy, J. F. M. and R. T. Haftka (1993). Approximation concepts for optimum structural design—a review. *Structural and Multidisciplinary Optimization 5*(3), 129–144. 29

Bendsøe, M. P. (1989). Optimal shape design as a material distribution problem. *Structural and Multidisciplinary Optimization 1*(4), 193–202. 33

Bendsøe, M. P., A. Ben-Tal, and J. Zowe (1994). Optimization methods for truss geometry and topology design. *Structural and Multidisciplinary Optimization 7*(3), 141–159. 34

Bhowmick, S., B. Toth, and P. Raghavan (2009). Towards low-cost, high-accuracy classifiers for linear solver selection. *Computational Science—ICSS 2009 5544*, 463–472. 41

Borkowski, A., S. Jendo, W. Prager, and M. Save (1990). *Structural Optimization: Volume 2: Mathematical Programming*, Volume 2. New York, NY: Springer. 28

Box, M. J. (1965). A new method of constrained optimization and a comparison with other methods. *The Computer Journal 8*(1), 42–52. 65

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 532. 129

Breiman, L., R. Olshen, J. Friedman, and C. Stone (1984). *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks. 129

Brent, R. P. (2002). *Algorithms for minimization without derivatives*. Mineola. NY: Dover Publications. 20, 64, 66, 98, 122

Brotchie, J. F. (1969). A criterion for optimal design of plates. In *ACI Journal Proceedings*, Volume 66. 37

California Department of Transport (2012). Bay bridge info. http://baybridgeinfo.org/. 78

Carmichael, D. G. (1981). *Structural modelling and optimization: a general methodology for engineering and control*. New York, NY: Ellis Horwood. 28

Caso, C. and M. Angeles gil (1988). The gini-simpson index of diversity: estimation in the stratified sampling. *Communications in Statistics—Theory and Methods 17*(9), 2981–2995. 129

Chan, A. S. L. (1960). The design of michell optimum structures. Technical Report 142, Cranfield College of Aeronautics. 28

Chang, C.-C. and C.-J. Lin (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST) 2*(3), 27. 130, 132

Chern, J. M. and W. Prager (1970). Optimal design of beams for prescribed compliance under alternative loads. *Journal of Optimization Theory and Applications 5*(6), 424–431. 36

Clune, R., J. J. Connor, and J. A. Ochsendorf (2011, June). Interactive structural optimization. In *Sixth M.I.T. Conference on Computational Fluid and Solid Mechanics*, Cambridge, MA. 15, 31, 151

Clune, R., J. J. Connor, J. A. Ochsendorf, and D. Kelliher (2012, June). An object-oriented architecture for extensible structural design software. *Computers and Structures 100-101*, 1–17. 13, 14, 15, 151

Cohn, M. Z. (1994). Theory and practice of structural optimization. *Structural and Multidisciplinary Optimization 7*(1), 20–31. 15, 16

Cohn, M. Z. and A. S. Dinovitzer (1994). Application of structural optimization. *Journal of Structural Engineering 120*(2), 617–650. 32, 36

Colgan, L., R. Spence, and P. Rankin (1995). The cockpit metaphor. *Behaviour and Information Technology 14*(4), 251–263. 31, 151

Conn, A. R., N. I. M. Gould, and P. L. Toint (1991). A globally convergent augmented langrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis 28*(2), 545–572. 20, 64

Czyzyk, J., M. P. Mesnier, and J. J. Mor (1998). The NEOS server. *Computing in Science and Engineering 5*(3), 68–75. 42

Davis, T. A. (2005, December). Algorithm 849: A concise sparse cholesky factorization package. *ACM Transactions on Mathematical Software 31*(4), 587591. 69

Deb, K. (1991). Optimal design of a welded beam via genetic algorithms. *AIAA journal 29*(11), 2013–2015. 36

Dongarra, J., G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, and K. Seymour (2006). Self-adapting numerical software (SANS) effort. *IBM Journal of Research and Development 50*(2.3), 223–238. 41

Dorn, W. S., R. E. Gomory, and H. J. Greenberg (1964). Automatic design of optimal structures. *Journal de Mecanique 3*(6), 25–52. 28

Duckstein, L. (1981). Multiobjective optimization in structural design: The model choice problem. Technical report, Defense Technical Information Center. 35

ECMA International (2013). Standard ECMA-372: C++/CLI language specification. http://www.ecma-international.org/publications/standards/Ecma-372.htm. 70

Eiben, A. E. and M. Jelasity (2002). A critical note on experimental research methodology in EC. In *CEC'02. Proceedings of the 2002 Congress on Evolutionary Computation.*, Volume 1, pp. 582–587. 16

Eijkhout, V. and E. Fuentes (2010). Machine learning for multi-stage selection of numerical methods. *New Advances in Machine Learning, INTECH*, 117–136. 41

Ewald, R., J. Himmelspach, and A. Uhrmacher (2008). An algorithm selection approach for simulation systems. In *22nd Workshop on Principles of Advanced and Distributed Simulation, 2008. PADS '08*, pp. 91–98. 41

Farkas, J. and M. N. Pavlovic (1984). *Optimum design of metal structures.* New York, NY: Ellis Horwood. 28

Flager, F., B. Welle, P. Bansal, G. Soremekun, and J. Haymaker (2009). Multidisciplinary process integration and design optimization of a classroom building. *Journal of Information Technology in Construction 14*, 595–612. 12, 13, 30, 31

Fourer, B. and D. Orban (2012). Dr. Ampl. http://www.gerad.ca/~orban/drampl/index.html. 42

Fourer, R., D. Gay, and B. Kernighan (2002). *AMPL: A Modeling Language for Mathematical Programming* (2nd ed.). Pacific Grove, CA: Brooks/Cole Publishing Company. 42

Fox, R. L. (1971). *Optimization methods for engineering design.* Reading, MA: Addison-Wesley. 28

Freund, Y. and R. E. Schapire (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 148–156. 129

Fu, G. and D. M. Frangopol (1988). Reliability-based multiobjective structural optimization, phase 2: Applications to frame systems. *Structural Research Series 88*(1). 30

Fuentes, E. (2007). *Statistical and machine learning techniques applied to algorithm selection for solving sparse linear systems.* Ph. D. thesis, University of Tennessee, Knoxville. 41

Fukunaga, A. S., S. Chien, D. Mutz, R. L. Sherwood, and A. D. Stechert (1997). Automating the process of optimization in spacecraft design. In *Aerospace Conference, 1997. Proceedings., IEEE*, Volume 4, pp. 411–427. 42

Gent, I. P., S. A. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. M. Smith, and T. Walsh (1997). How not to do it. Technical report, University of Leeds School of Computer Studies. 16

Gomes, H. M. (2011). Truss optimization with dynamic constraints using a particle swarm algorithm. *Expert Systems with Applications 38*(1), 957–968. 34

Guan, H., Y. J. Chen, Y. C. Loo, Y. M. Xie, and G. P. Steven (2003). Bridge topology optimisation with stress, displacement and frequency constraints. *Computers and Structures 81*(3), 131–145. 36

Guo, T., A. Li, and H. Wang (2007a). Application comparison between zero-order and first-order optimization methods in model updating of suspension bridges. *Journal of Vibration and Shock 4*, 8. 35

Guo, T., A. Li, and H. Wang (2007b). Research in form-finding of suspension structures based on Newton-Raphson iteration and zero order optimization arithmetic. *Engineering Mechanics 4*. 35

Haftka, R. T. and Z. Gürdal (1992). *Elements of structural optimization*, Volume 11. Dordrecht, The Netherlands: Kluwer Academic Publishers. 12, 15, 28

Hagan, M. T. and M. B. Menhaj (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks 5*(6), 989–993. 130

Hajela, P. and E. Lee (1995). Genetic algorithms in truss topological optimization. *International Journal of Solids and Structures 32*(22), 3341–3357. 12, 34

Halkidi, M., Y. Batistakis, and M. Vazirgiannis (2001). On clustering validation techniques. *Journal of Intelligent Information Systems 17*(2), 107–145. 78

Hastie, T., R. Tibshirani, J. Friedman, and J. Franklin (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer 27*(2), 83–85. 127, 128, 129, 133, 136

Heege, M. (2007, April). *Expert Visual C++/CLI: .NET for Visual C++ Programmers.* New York, NY: Springer-Verlag. 70

Hemp, W. S. (1973). *Optimum structures.* United Kingdom: Clarendon Press. 28

Heyman, J. (1959). On the absolute minimum weight design of framed structures. *The Quarterly Journal of Mechanics and Applied Mathematics 12*(3), 314–324. 28

Hilario, M., A. Kalousis, P. Nguyen, and A. Woznica (2009). A data mining ontology for algorithm selection and meta-mining. In *Proceedings of the ECML/PKDD09 Workshop on 3rd Generation Data Mining*, pp. 76–87. 41

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press. 29

Hough, P. D. and P. J. Williams (2006). Modern machine learning for automatic optimization algorithm selection. In *Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop.* 42

Houstis, E. N., A. C. Catlin, J. R. Rice, V. S. Verykios, N. Ramakrishnan, and C. E. Houstis (2000). PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software 26*(2), 227–253. 39, 40

Hubka, R. E. (1968). Structural optimization of six different types of rectangular plates subjected to combined and biaxial-compressive loading. Technical Report 21662, Lockheed-California Company. 37

Imai, K. and L. A. Schmit (1981). Configuration optimization of trusses. *Journal of the Structural Division 107*(5), 745–756. 32, 33

Jendo, S. and W. Marks (1986, January). Multiobjective structural optimization. In A. Prkopa, J. Szelezsan, and B. Strazicky (Eds.), *System Modelling and Optimization*, Number 84 in Lecture Notes in Control and Information Sciences, pp. 365–374. Springer Berlin Heidelberg. 30

Johnson, S. G. (2012). The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt. 20, 64, 70, 168

Jolliffe, I. T. (2002). *Principal component analysis*. New York, NY: Springer-Verlag. 135

Jones, D. R., C. D. Perttunen, and B. E. Stuckman (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications 79*(1), 157–181. 20, 64, 66, 98, 122

Kaelo, P. and M. M. Ali (2006). Some variants of the controlled random search algorithm for global optimization. *Journal of optimization theory and applications 130*(2), 253–264. 20, 64, 66, 98, 123

Kapoor, M. P. and K. Kumarasamy (1981). Optimum configuration of transmission towers in dynamic response regime. In *Proceedings of the International Symposium on Optimum Structural Design*, pp. 185–189. 34

Kathiravan, R. and R. Ganguli (2007). Strength design of composite beam using gradient and particle swarm optimization. *Composite Structures 81*(4), 471–479. 36

Kent, W. (1983). A simple guide to five normal forms in relational database theory. *Communications of the ACM 26*(2), 120–125. 71

Kirkpatrick, S. and M. P. Vecchi (1983). Optimization by simmulated annealing. *Science 220*(4598), 671–680. 29

Kirsch, U. (1993). *Structural Optimization: Fundamentals and Applications* (1st ed.). New York, NY: Springer. 28

Kruskal, W. H. and W. A. Wallis (1952). Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association 47*(260), 583–621. 97

Lin, J. H., W. Y. Che, and Y. S. Yu (1982). Structural optimization on geometrical configuration and element sizing with statical and dynamical constraints. *Computers and Structures 15*(5), 507–515. 34

Lingyun, W., Z. Mei, W. Guangming, and M. Guang (2005). Truss optimization on shape and sizing with frequency constraints based on genetic algorithm. *Computational Mechanics 35*(5), 361–368. 34

Liu, Y., Z. Li, H. Xiong, X. Gao, and J. Wu (2010). Understanding of internal clustering validation measures. In *2010 IEEE 10th International Conference on Data Mining*, pp. 911–916. 80

Lounis, Z. and M. Z. Cohn (1995). An engineering approach to multicriteria optimization of bridge structures. *Computer-Aided Civil and Infrastructure Engineering 10*(4), 233–238. 34, 35

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Volume 1, pp. 14. 79

Mahfoud, S. (1995). *Niching Methods for Genetic Algorithms.* Ph. D. thesis, University of Illinois at Urbana Champaign, Urbana, Illinois. 31

Martini, K. (2011). Harmony search method for multimodal size, shape, and topology optimization of structural frameworks. *Journal of Structural Engineering 137*(11), 1332–1339. 31, 35, 36

Masur, E. F. (1970). Optimum stiffness and strength of elastic structures. *Journal of the Engineering Mechanics Division 96*(5), 621–640. 37

MATLAB (2012). *version 8.8.0.783 (R2012b).* Natick, Massachusetts: The Mathworks Inc. 129

Michell, A. G. (1904). The limits of economy of material in frame structures. *Philosophical Magazine 8*(47), 589–597. 12, 27, 28

Microsoft (2013a). LINQ to SQL. http://msdn.microsoft.com/en-us/library/bb386976.aspx. 71

Microsoft (2013b). Microsoft .NET framework. http://www.microsoft.com/net. 69

Microsoft Technet (2013). SQL Server 2008 R2. http://technet.microsoft.com/en-us/sqlserver/ff398089.aspx. 71

Mittelmann, H. and P. Spellucci (1998). Decision tree for optimization software. http://plato.asu.edu/guide.html. 42

Nelder, J. A. and R. Mead (1965). A simplex method for function minimization. *The Computer Journal 7*(4), 308–313. 20, 28, 64, 65, 98, 121

Oasys (2012). GSA suite. http://www.oasys-software.com/products/engineering/gsa-suite.html. 18, 68, 168

Ohsaki, M. and C. C. Swan (2002). Topology and geometry optimization of trusses and frames. In S. A. Burns (Ed.), *Recent Advances in Optimal Structural Design*, pp. 97–123. 33

Ong, Y. and A. Keane (2002). A domain knowledge based search advisor for design problem solving environments. *Engineering Applications of Artificial Intelligence 15*(1), 105–116. 42

Pedersen, C. B. W. and P. Allinger (2006, January). Industrial implementation and applications of topology optimization and future needs. In M. P. Bendsøe, N. Olhoff, and O. Sigmund (Eds.), *IUTAM Symposium on Topological Design Optimization of Structures, Machines and Materials*, Solid Mechanics and Its Applications, pp. 229–238. Springer Netherlands. 33, 34

Powell, M. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J.-P. Hennart (Eds.), *Advances in Optimization and Numerical Analysis*. Dordrecht: Kluwer Academic. 20, 64, 65, 98, 120

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal 7*(2), 155–162. 28, 66

Powell, M. J. D. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report Cambridge NA Report NA2009/06, University of Cambridge, UK. 20, 64, 65, 98, 120

Prager, W. and G. I. N. Rozvany (1977). Optimal layout of grillages. *Journal of Structural Mechanics 5*(1), 1–18. 28

Pylon Lookout (2003). Pylon lookout. http://www.pylonlookout.com.au/FrameTOP.htm. 77

Rahmatalla, S. and C. C. Swan (2003). Form finding of sparse structures with continuum topology optimization. *Journal of Structural Engineering 129*(12), 1707–1716. 36

Rajeev, S. and C. S. Krishnamoorthy (1997). Genetic algorithms-based methodologies for design optimization of trusses. *Journal of Structural Engineering 123*(3), 350–358. 34

Rao, S. S. (1987). Game theory approach for multiobjective structural optimization. *Computers and Structures 25*(1), 119–127. 30

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers 15*, 65–118. 39, 40, 41

Rice, J. R. (1979). Methodology for the algorithm selection problem. In *Proceedings of the IFIP TC 2.5 Working Conference on Performance Evaluation of Numerical Software*, pp. 301–307. 39

Rowan, T. H. (1990). *Functional Stability Analysis Of Numerical Algorithms.* Ph. D. thesis, University of Texas at Austiin, Austin, TX. 20, 64, 65, 98, 121

Rozvany, G. I. N. (1998). Exact analytical solutions for some popular benchmark problems in topology optimization. *Structural and Multidisciplinary Optimization 15*(1), 42–48. 16, 27

Rozvany, G. I. N. (2009). A critical review of established methods of structural topology optimization. *Structural and Multidisciplinary Optimization 37*(3), 217–237. 12, 33, 34

Rozvany, G. I. N. and W. Prager (1989). *Structural design via optimality criteria: the Prager approach to structural optimization*, Volume 8. Dordrecht, The Netherlands: Kluwer Academic Publisher. 12, 27, 28

Runarsson, T. P. and X. Yao (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35*(2), 233–243. 20, 64, 66, 98, 123

Save, M., W. Prager, G. Sacchi, and W. H. Warner (1985). *Structural Optimization: Optimality criteria*, Volume 1. New York, NY: Plenum Publishing Corporation. 28

Schmit, L. A. and T. P. Kicher (1962). Synthesis of material and configuration selection. *Journal of the Structural Division 88*(3), 79–102. 32

Shea, K. and J. Cagan (1999). Languages and semantics of grammatical discrete structures. *AI EDAM 13*(4), 241 251. 31

Sheu, C. Y. and W. Prager (1968). Minimum-weight design with piecewise constant specific stiffness. *Journal of Optimization Theory and Applications 2*(3), 179–186. 36

Siddall, J. N. (1982). *Optimal engineering design: principles and applications*, Volume 14. New York, NY: Marcel Dekker. 28

Smith, J., J. Hodgins, I. Oppenheim, and A. Witkin (2002). Creating models of truss structures with optimization. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, New York, NY, USA, pp. 295–301. ACM. 13

Smith-Miles, K. A. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys 41*(1), 1–25. 39, 40, 128

Spearman, C. (1904). The proof and measurement of association between two things. *The American journal of psychology 15*(1), 72–101. 97

Spillers, W. R. and K. M. MacBain (2009). *Structural Optimization* (2009 ed.). New York, NY: Springer. 28

Topping, B. H. V. (1983). Shape optimization of skeletal structures: a review. *Journal of Structural Engineering 109*(8), 1933–1951. 12, 32

Vanderplaats, G. N. (1984). *Numerical optimization techniques for engineering design: with applications*, Volume 32. New York, NY: McGraw-Hill. 15, 28

Vapnik, V. (1999). *The nature of statistical learning theory.* Springer. 132

Venkayya, V. B. (1978). Structural optimization: a review and some recommendations. *International Journal for Numerical Methods in Engineering 13*(2), 203–228. 12

Von Buelow, P. (2008). Suitability of genetic based exploration in the creative design process. *Digital Creativity 19*(1), 51. 15, 31, 151

Weerawarana, S., E. N. Houstis, J. R. Rice, A. Joshi, and C. E. Houstis (1996). PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Transactions on Mathematical Software 22*(4), 447–468. 40

Wilson, D. C., D. B. Leake, and R. Bramley (2000). Case-based recommender components for scientific problem-solving environments. In *Proceedings of the Sixteenth International Association for Mathematics and Computers in Simulation World Congress.* 41

Wolpert, D. and W. Macready (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*(1), 67–82. 18

Xie, Y. M. and G. P. Steven (1993). A simple evolutionary procedure for structural optimization. *Computers and Structures 49*(5), 885–896. 33

Appendix A

Details of performance variation study

This appendix provides additional details of the structural design problems, results, and computational setup from §1.2's empirical investigation of algorithm performance. A full description of the algorithms is given in Chapter 3, §3.2.1.

A.1 Design problems

Before describing each design problems in detail, this section presents a few design criteria common to all three.

They all use Load and Resistance Factor Design criteria [American Institute of Steel Construction, 2011]. We evaluate all stress constraints on structural elements using a load combination of 1.2 times dead load plus 1.6 times live load, and evaluate displacement constraints using only un-factored live loads. Load magnitudes are specified for each problem in the following sections. We assume material densities of $7850\,\mathrm{kg/m^3}$ for steel and $2400\,\mathrm{kg/m^3}$ for concrete throughout.

To simplify the problem of element sizing, element cross-sectional areas are chosen as continuous optimization design variables. Major, minor, and torsional moments of inertia depend on cross-sectional area according to Eq. A.1, the result of fitting linear least-squares regressions (through zero) between each of these sectional properties and the cross-sectional areas over all wide-flange sections in the American Institute of Steel Construction's steel tables [American Institute of Steel Construction, 2011]. This is a simplification of the piece-wise linear relationships in Chapter 3, but is sufficient for the

purposes of this initial investigation.

$$\begin{aligned} I_{maj}(A) &= 1.249 \times 10^{-1} A \\ I_{min}(A) &= 7.496 \times 10^{-3} A \\ J(A) &= 7.891 \times 10^{-4} A \end{aligned} \tag{A.1}$$

where $I_{maj}(A)$, $I_{min}(A)$, and $J(A)$ are the major, minor, and torsional moments of inertia, expressed as linear functions of the cross-sectional area, A.

Mass minimization of a steel arch bridge by varying member sizing and geometry

The first design problem is the structural mass minimization of a simply-supported steel arch bridge which spans 30.5 m (Fig. A.1). The bridge, in addition to its own weight, supports a dead load of 3.80 kN/m^2 and a pedestrian load of 4.10 kN/m^2, both distributed uniformly along the deck.

The design variables are the cross-sectional areas of the longitudinal deck beams (A_1), the transverse deck beams (A_2), the hangers (A_3), the main arch ribs (A_4), and the struts that link the two arches (A_5). The major, minor, and torsional moments of inertia are set according to Eq. A.1. The span-to-height ratio of the parabolic arches, γ, is a geometric

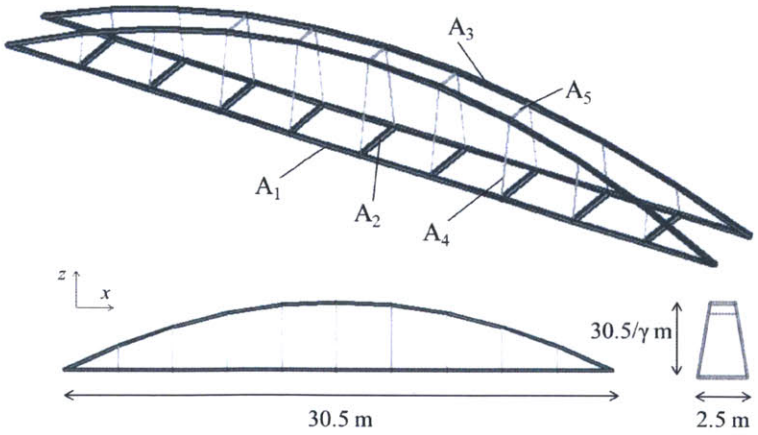


Figure A.1 – The starting point for steel arch bridge design, with a span-to-height ratio, γ, of 8.20

design variable, controlling overall structural geometry according to Eqs. A.2 and A.3.

$$z(x,\gamma) = \frac{(x - \frac{L}{2})^2}{L(\frac{1}{\gamma} - \frac{\gamma}{4})} \tag{A.2}$$

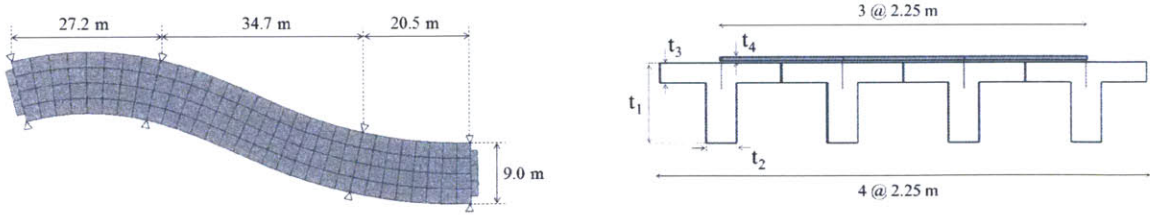
$$\Delta y(z,\gamma) = z\frac{W\gamma}{4L} \tag{A.3}$$

where x and z are the x- and z-coordinates of points along the arch (both measured from the left-hand intersection of the arch and the deck), L is the span of the arch (30.5 m), W is the width of the deck (2.5 m), and Δy is the magnitude of the inward lean of the arch at a given height z. Table A.1 shows the initial values of all design variables, along with their upper and lower bounds.

All nodal displacements have an upper bound of 0.04 m, the stress in deck and hanger elements is limited to 290 $\mathrm{MN/m^2}$, and stresses in the main arch are bounded above by 310 $\mathrm{MN/m^2}$.

Table A.1 – Design variable initial values and bounds for the steel arch bridge

Design variable (units)	Lower bound	Initial value	Upper bound
A_1 (10^{-3} $\mathrm{m^2}$)	0.65	1.29	12.90
A_2 (10^{-3} $\mathrm{m^2}$)	0.80	1.59	15.90
A_3 (10^{-2} $\mathrm{m^2}$)	0.70	1.39	13.90
A_4 (10^{-3} $\mathrm{m^2}$)	2.81	5.61	56.10
A_5 ($\mathrm{m^2}$)	0.29	1.15	11.50
γ	4.0	8.2	12.0



(a) (b)

Figure A.2 – Concrete footbridge **(a)** in plan and **(b)** in section, showing the four sizing design variables, t_1, t_2, t_3, and t_4

Mass minimization of concrete footbridge

The bridge shown in Fig. A.2 is composed of four longitudinal concrete T-beams which support a one-way spanning concrete deck and are themselves simply supported at locations shown by triangles. The design loads are the weight of the bridge and a uniformly-distributed pedestrian live load of $4.10\,\mathrm{kN/m^2}$.

The four design variables are the overall depth and web thickness of the T-beams, t_1 and t_2, the flange thickness of the identically-sized T-beams, t_3, and the thickness of the concrete slab, t_4 (Table A.2 shows their initial values and bounds.) Nodal displacements are constrained to be less than 0.052 m, element stresses are constrained to be less than $25\,\mathrm{MN/m^2}$, and the objective function for optimization is the structural mass.

Initial investigations show the design space to be reasonably smooth and well-behaved, although this absence of noise and steep local gradients does not prohibit the existence of other features – such as multiple local optima – which may cause difficulty for some algorithms.

Table A.2 – Design variable initial values and bounds for concrete footbridge

Design variable (units)	Lower bound	Initial value	Upper bound
t_1 (m)	0.50	2.00	4.00
t_2 (m)	0.10	0.50	1.00
t_3 (m)	0.10	0.75	1.00
t_4 (m)	0.10	0.10	1.00

Mass minimization of suspension bridge

This design problem seeks the minimum structural mass of the central 800 m span of a suspension bridge, shown in Fig. A.3. The presence of the back-spans and bridge towers is represented by fully restraining all displacements and rotations at the ends of the cables. The simply-supported bridge deck carries a dead load of $3.2\,\mathrm{kN/m^2}$ in addition to its self weight. The deck supports four lanes of traffic, represented as a uniformly-distributed live load of $3.6\,\mathrm{kN/m^2}$.

The sizing design variables are the cross-sectional areas of the outer and inner longitudinal deck beams (A_1 and A_2), the transverse deck beams (A_3), the hangers (A_4), and the cables (A_5). The location of the deck remains fixed in space, with a constant distance of 3 m between the neutral axis of the longitudinal deck beams and the vertex of the parabola that defines the cable geometry. We include the distance from the parabolas vertex to its

focus, p, as a geometric design variable which controls geometry according to Eq. A.4.

$$z(x,p) = \frac{(x - \frac{L}{2})^2}{4p} + 3 \tag{A.4}$$

where x and z are the x- and z-coordinates of a cable node (both measured from the end of the deck), and L is the bridge span.

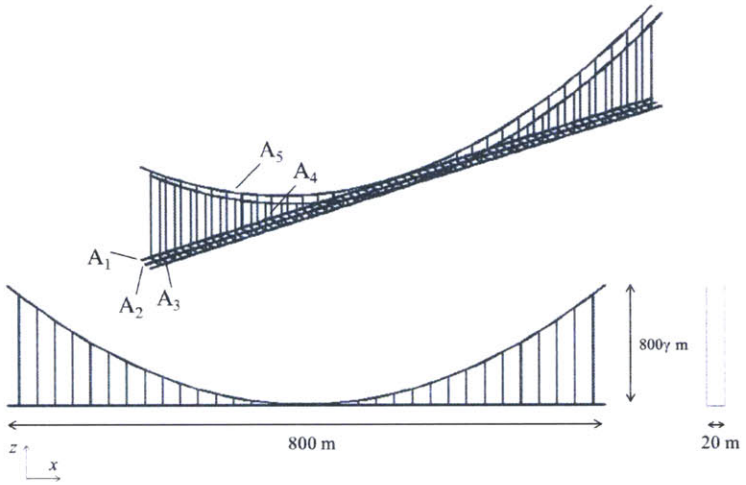


Figure A.3 – The structural behavior of the central span of a suspension bridge is sensitive to small changes in geometry.

All nodal displacements are constrained to be less than 1.0 m, and element stresses to be less than 300 MN/m^2. To accurately predict the structural behavior of the long-span cables, the bridge model accounts for material and geometric nonlinearities. This behavior and the associated values of the objective and constraint functions are highly sensitive to changes in geometry, leading to steep local gradients which may cause difficulty for some algorithms. Table A.3 shows initial values of, and bounds on, the design variables.

Table A.3 – Design variable initial values and bounds for suspension bridge

Design variable (units)	Lower bound	Initial value	Upper bound
A_1 (m^2)	0.01	0.50	5.00
A_2 (m^2)	0.01	0.10	5.00
A_3 (m^2)	0.01	0.50	5.00
A_4 (m^2)	0.05	0.80	5.00
A_5 (m^2)	0.01	0.05	5.00
p (m)	200	540	1000

A.2 Software and hardware specifications

Structural analyses, both linear and non-linear, are performed using a custom application programming interface (described in Chapter 3) to the *Oasys GSA* structural analysis package [Oasys, 2012]. We use version 2.2.3 of the *NLopt* optimization library, wrapped for the *.NET* environment as described in §3.3.2 [Johnson, 2012]. Source code for this wrapper, and the underlying *NLopt* library, are available at `https://github.com/roryclune/NLoptDotNet`. All relevant data is stored using the schema described in §3.3.3.

Computation occurs on a Dell *PowerEdge*® server running the 32-bit version of *Windows Server*® *Web 2007*. The server is equipped with two *Intel*® *Xeon*™ 3.20 GHz CPUs and 6.00 GB of memory.

A.3 Detailed results

Tables A.4, A.5, and A.6 show the design variable values on which each of the algorithms converge when solving the design problems, the mass of the resulting bridge designs, and the required number of calls to the analysis software. Figs. A.4, A.5, and A.6 show the best designs—in terms of minimum objective value—attained for each of the three problems.

Fig. A.4 shows the best quality (lowest mass) design achieved for the design of the steel arch bridge. COBYLA and SUBPLEX performed best in terms of design qual-

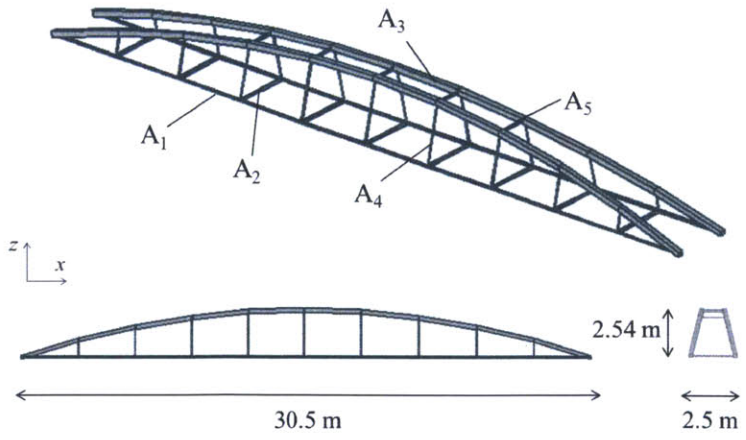


Figure A.4 – COBYLA and SUBPLEX achieved the lowest structural mass for the arch bridge, substantially reducing its depth.

Table A.4 – Final design variables and performance measures for all algorithms for the steel arch bridge problem

Algorithm	A_1 (10^{-4}m^2)	A_2 (10^{-4}m^2)	A_3 (10^{-3}m^2)	A_4 (10^{-4}m^3)	A_5 (10^{-4}m^2)	γ	Analysis calls	Mass (10^3kg)
COBYLA	6.45	7.95	6.95	2.81	5.75	12.00	143	3.94
BOBYQA	6.45	7.95	6.95	2.81	5.75	11.36	16	3.95
CRS	6.85	7.95	6.95	3.02	7.37	10.04	45	4.02
SUBPLEX	6.45	7.95	6.95	2.81	5.75	12.00	57	3.94
PR-AXIS	6.52	11.70	1.17	3.00	5.76	8.95	60	6.48
DIRECT	6.88	10.91	6.98	3.42	8.22	8.87	3,422	5.73
CRS	6.92	7.95	5.95	3.35	8.42	9.95	3,441	4.15
ISRES	6.98	13.00	8.52	3.04	7.41	10.02	2,783	4.85
Starting point	12.90	15.90	13.90	0.56	11.50	8.20	N/A	8.09

ity, with both converging on the maximum possible span-to-height ratio (i.e., the lowest arch height) of 12.

BOBYQA performed almost as well as COBYLA and SUBPLEX in terms of design quality, but required dramatically fewer analysis software calls. All algorithms converged to answers that improve on the starting point, although quality of these answers is quite diverse (Table A.4).

Fig. A.5 shows a cross-sectional view of the lightest design for the concrete footbridge, achieved by both NEL-MEAD and CRS, with NEL-MEAD making an order of magnitude fewer analysis calls. These algorithms converged on a design with greater web depth in the concrete T-sections than the starting point (Table A.5), which is a more efficient use of material in bending than in many of the other resulting designs.

The non-linear structural analyses of the suspension bridge take a relatively long time to

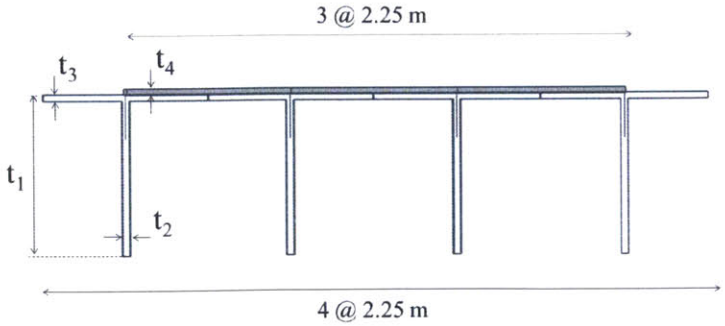


Figure A.5 – CRS and NEL-MEAD, followed closely by four others, produced the lightest concrete bridges.

Table A.5 – Final design variables and performance measures for all algorithms on the concrete footbridge problem

Algorithm	t_1 (m)	t_2 (m)	t_3 (m)	t_4 (m)	Analysis calls	Mass (10^6 kg)
COBYLA	2.464	0.153	0.100	0.118	54	0.82
BOBYQA	2.029	0.430	0.402	0.101	23	1.87
NEL-MEAD	2.930	0.100	0.100	0.123	230	0.76
SUBPLEX	2.132	0.215	0.100	0.100	505	0.84
PR-AXIS	1.554	0.570	0.101	0.115	179	1.22
DIRECT	2.833	0.111	0.100	0.133	10,226	0.8
CRS	2.926	0.100	0.100	0.123	2,938	0.76
ISRES	2.941	0.131	0.113	0.121	2,952	0.86
Starting point	2.000	0.500	0.750	0.100	N/A	2.49

run, especially in the case of intermediate designs which undergo large displacements. Because of this, algorithms cannot evaluate as many designs as in the previous two problems within the six hour time limit.

NEL-MEAD again achieves the result with the lowest mass (Fig. A.6), although it requires very many function evaluations to do so. With reference to Eq. A.4, the height-to-span ratio of the cable in this case is 0.17. This is the deepest of any of the eight cable designs,

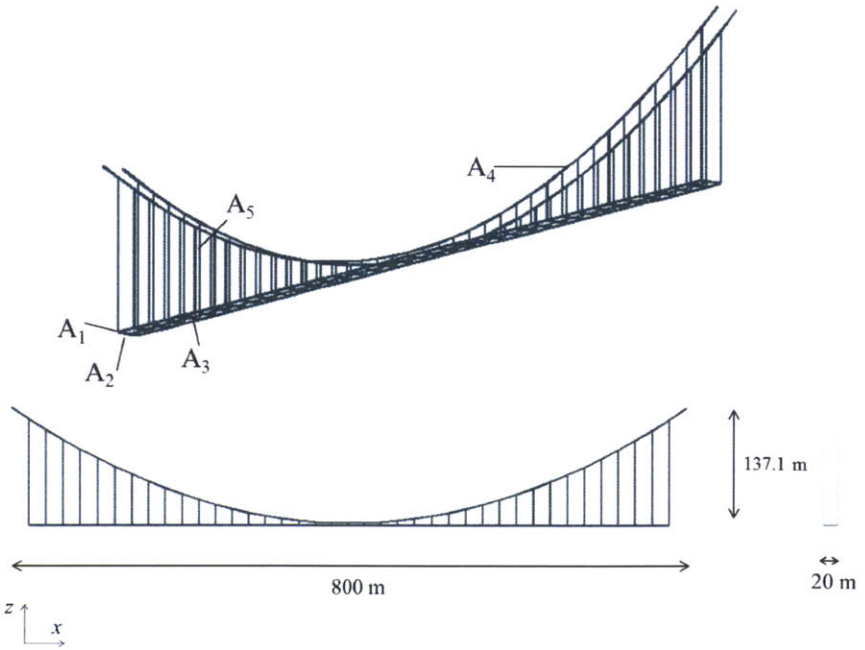


Figure A.6 – Despite requiring many analysis calls to converge, Nelder-Mead (algorithm 3) outperformed all other algorithms in terms of objective value

but the increased cable stiffness allows for less material per unit length in the structural elements, ultimately resulting in a lighter design.

As discussed in Chapter 1, these results demonstrate substantial variation in the performance of algorithms, and the pattern of this performance varies from problem to problem. They demonstrate clear benefit, both in terms of design quality and computational cost, to selecting the appropriate algorithm, and further motivate the development of schemes to aid and to automate this selection.

Table A.6 – Final design variables and performance measures for all algorithms on the suspension bridge problem

Algorithm	A_1 (m^2)	A_2 (m^2)	A_3 (m^2)	A_4 (m^2)	A_5 (m^2)	p (m)	Analysis calls	Mass (10^7 kg)
COBYLA	0.01	0.02	0.63	0.44	0.03	411.1	56	1.06
BOBYQA	0.47	0.09	0.02	0.74	0.05	499.1	31	1.43
NEL-MEAD	0.01	0.06	0.01	0.23	0.01	291.7	300	0.43
SUBPLEX	0.01	0.01	0.01	0.41	0.01	553.3	174	0.57
PR-AXIS	0.08	0.01	0.35	0.73	0.03	659.4	106	1.24
DIRECT	0.84	0.29	0.84	1.98	0.29	600.0	370	4.31
CRS	0.01	0.01	0.01	0.93	0.01	944.9	333	1.21
ISRES	0.50	0.10	0.50	0.80	0.05	540.0	319	1.83
Starting point	0.50	0.10	0.50	0.80	0.05	540.0	N/A	1.83

Appendix B

Details of data generation method

This appendix provides supplementary details of Chapter 3's data generation method. §B.1 details the linear regressions used in §3.1.1's continuous approximation to the discrete element sizing problem, and further supports the assumption of approximate linearity that underpins it. §B.2 provides additional details of the structural models on which the design problems are based, and §B.3 concludes the appendix by expanding on the range of the experimental design.

B.1 Continuous approximation to discrete element sizing

§3.1.1 described a piecewise-linear (and, hence, piecewise-continuous) approximation to the discrete element sizing process prevalent in AISC code-based steel design [American Institute of Steel Construction, 2011]. In code-based design, engineers select a steel section with fixed sectional properties from a finite catalog and evaluate its physical performance. Our piecewise approximation allows optimization algorithms to size elements with a single continuous design variable, representing cross-sectional area; the other relevant sectional properties—major, minor, and torsional moments of inertia —become dependent variables.

The core of this approximation is a series of regressions between cross-sectional area and the three dependent sectional properties. Within each category of AISC-defined W-sections (grouped by their approximate height), these relationships are approximately linear, as explained in §3.1.1.

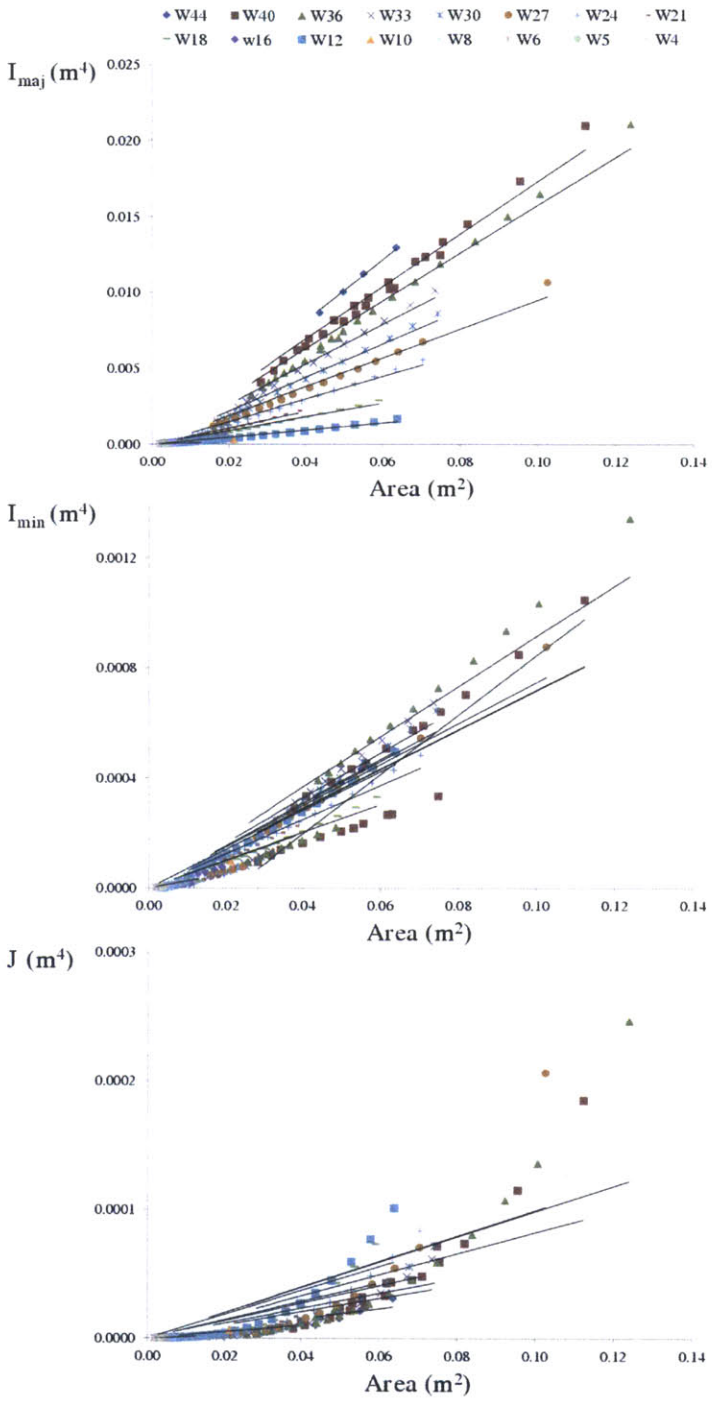


Figure B.1 – Plots of cross-sectional area against major, minor, and torsional area moments of inertia (I_{maj}, I_{min}, and J) for each AISC W-section category show the approximately linear relationships within section categories on which the continuous sizing method relies. Solid black lines show least-squares linear regressions through the origin for each category.

Taking advantage of this linearity, we fit least-squares linear regressions between area and each of major, minor, and torsional moments of inertia ($I_{\text{maj}}, I_{\text{min}}$, and J) for all W-section categories, using data from version 14 of the AISC shape catalog (Fig. B.1). To prevent dependent properties from taking on infeasible negative values for positive cross-sectional areas, each regression is forced to pass through the origin. The approximately linear relationships in the underlying data are evident in Fig. B.1.

The linear model coefficients from these regressions are used to size elements in optimization. An algorithm sets a value of cross-sectional area, and the chosen linear model is the one that maximizes I_{maj}; I_{min} and J are determined according to their linear models for the same W-section category. The regression coefficients are shown in Table B.1. (Since all models pass through the origin, they are defined by a single coefficient.)

Table B.1 – The linear model coefficients (Fig. B.1) area used to determine major, minor, and torsional moments of inertia (I_{maj}, I_{min}, and J) from the cross-sectional area, A.

W-section category	**Regression coefficients**		
	$\frac{I_{maj}}{A}$	$\frac{I_{min}}{A}$	$\frac{J}{A}$
	($\times 10^2$ m^2)	($\times 10^3$ m^2)	($\times 10^5$ m^2)
W44	20.23	7.78	38.10
W40	17.29	7.19	82.31
W36	15.73	9.15	98.79
W33	13.14	8.16	51.55
W30	10.99	7.66	58.25
W27	9.47	7.48	99.38
W24	7.43	6.20	68.25
W21	5.39	4.96	25.52
W18	4.51	5.04	71.49
W16	3.09	3.25	10.89
W12	2.34	7.04	92.84
W10	1.31	4.11	17.42
W8	0.83	2.56	9.93
W6	0.44	1.18	2.81
W5	0.30	1.05	3.23
W4	0.19	0.65	0.51

B.2 Remaining structural models

Chapter 3's experimental design is based on fourteen structural models of six different types—trussed arch, basket-handle arch, cable-stayed, girder, suspension, and Warren truss bridges. Within each type, the primary difference between models is their span length. The dissertation's main body presented a single example of each type; to aid reproduction of the study, this section presents details on the remaining eight bridge models. In all cases, these additional models' geometry is controlled by the same equations used in Chapter 3, referenced here for clarity. The starting values of their design variables are indicated in Tables 3.4 through 3.9.

The first structural type is the **basket-handle arch**, first introduced in §3.1.2. In addition to the previously-introduced model spanning 200 m, the experimental design (§3.1) contains models with 50 m and 300 m spans (Fig. B.2). Their arch geometry is controlled by Eq. 3.3, and they have the same element configuration as the 200 m model, apart from

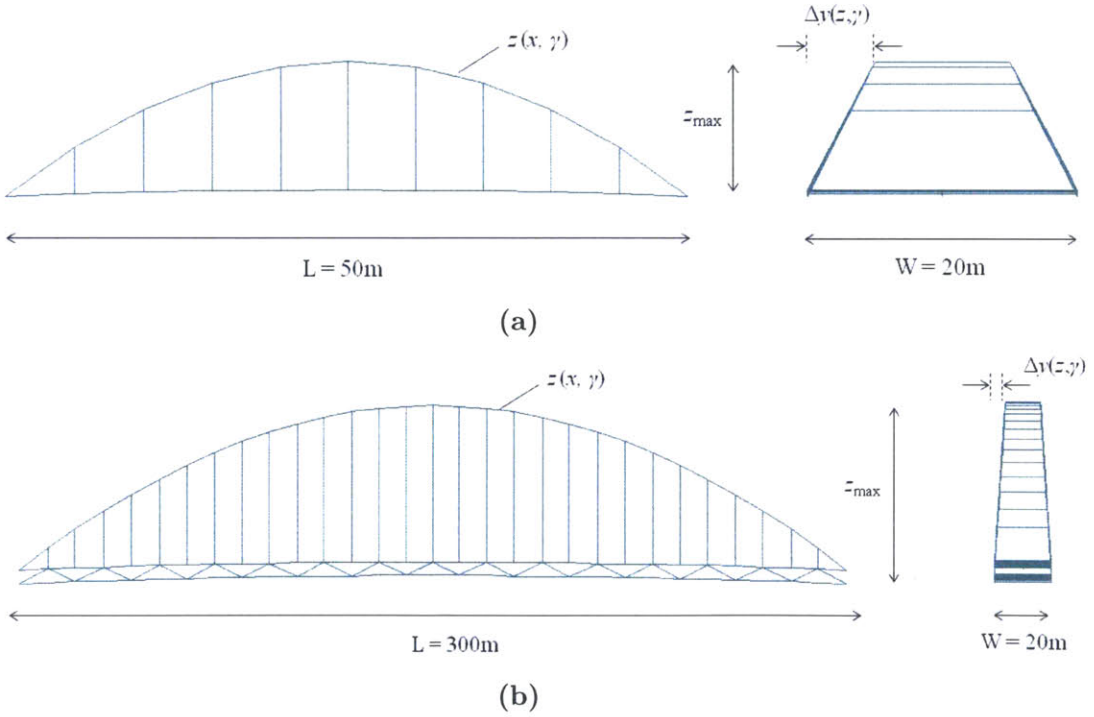


Figure B.2 – In addition to the 200 m steel backet-handle arch shown in the dissertation's main body (Fig. 3.4), the experimental design includes two other basket-handle arches of span lengths **(a)** 50 m and **(b)** 300 m. Each of these model's arch geometry is defined by Eq. 3.3. The 50 m arch bridge uses the single-layer deck instead of the double-layer truss deck used by the other two (see §3.1.2)

the 50 m version's use of a single-layer road deck (§3.1.2).

The additional **trussed arch** model has a clear span of 300 m (Fig. B.3); the geometry of its inner and outer arches in controlled by Eq. 3.4 and 3.5, and it uses the same element types and configuration as the 200 m version introduced in Chapter 3.

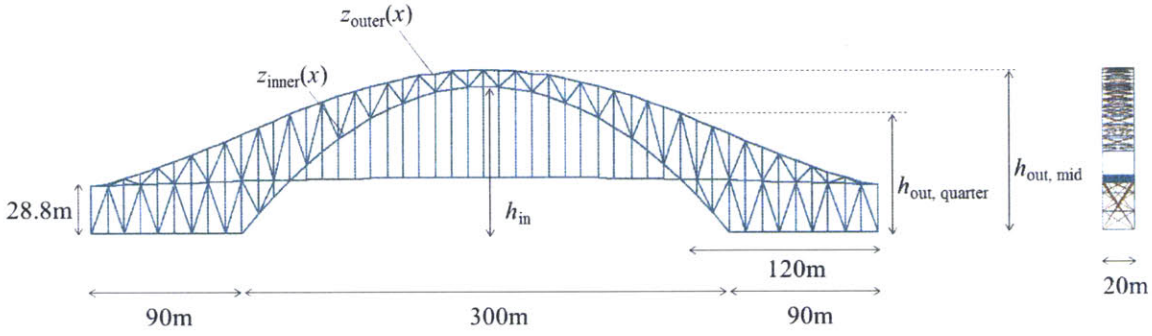
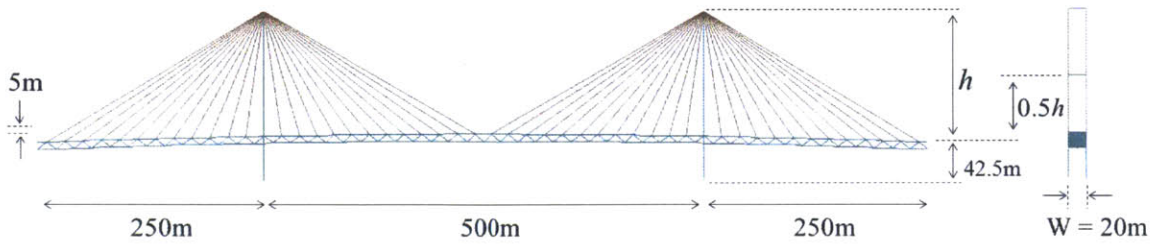


Figure B.3 – The geometry of the 300 m trussed arch bridge is defined by $z_{inner}(x)$ and $z_{outer}(x)$ in accordance with Eqs. 3.4 and 3.5. It is composed of the same types of structural elements as the 200 m trussed arch presented in §3.1.2.

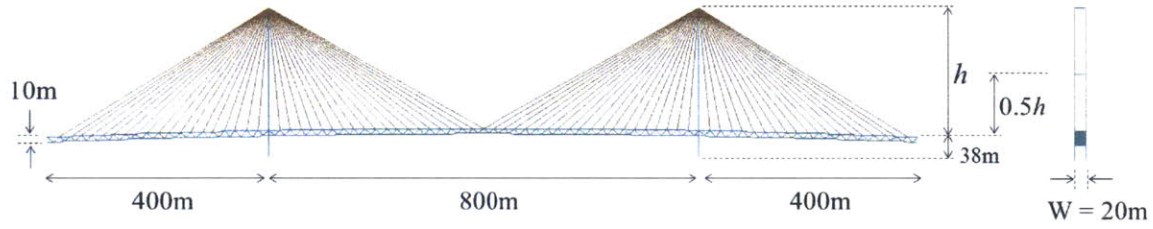
Two extra **cable-stayed bridge** models, spanning 500 m and 800 m, also have a single geometric design variable, h (Fig. B.4). Both are analyzed using the same geometrically-nonlinear simulation techniques used for the 300 m version, described in Chapter 3.

The 800 m-long **suspension bridge** model (Fig. B.5) behaves similarly to the 500 m version in Chapter 3. Its parabolic cable geometry is also controlled by Eq. 3.7.

The final category of bridges—the Warren trusses—consist of the 200 m-long model in Chapter 3 and the 50 m- and 300 m- long versions shown in Fig. B.6. All three use the same element types (Table 3.9) and have a single geometric design variable, h.



(a)



(b)

Figure B.4 – As in the case of the 300 m cable-stayed bridge in the dissertation's main body, the additional **(a)** 500 m and **(b)** 800 m models have a single geometric sizing variable, h, which determines the height of the towers.

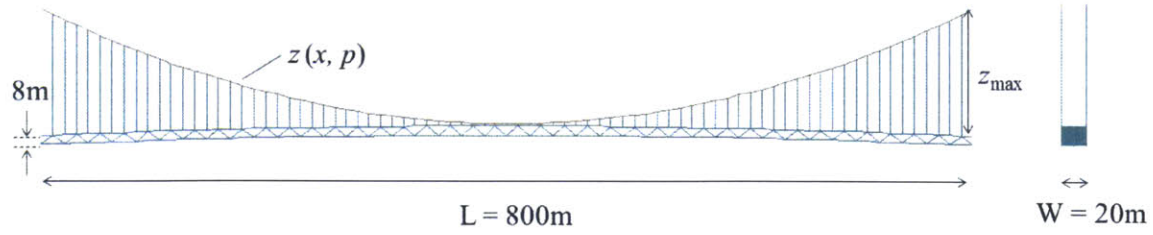


Figure B.5 – The geometry of the 800 msuspension bridge model is, like the 500 m model, controlled by Eq. 3.7.

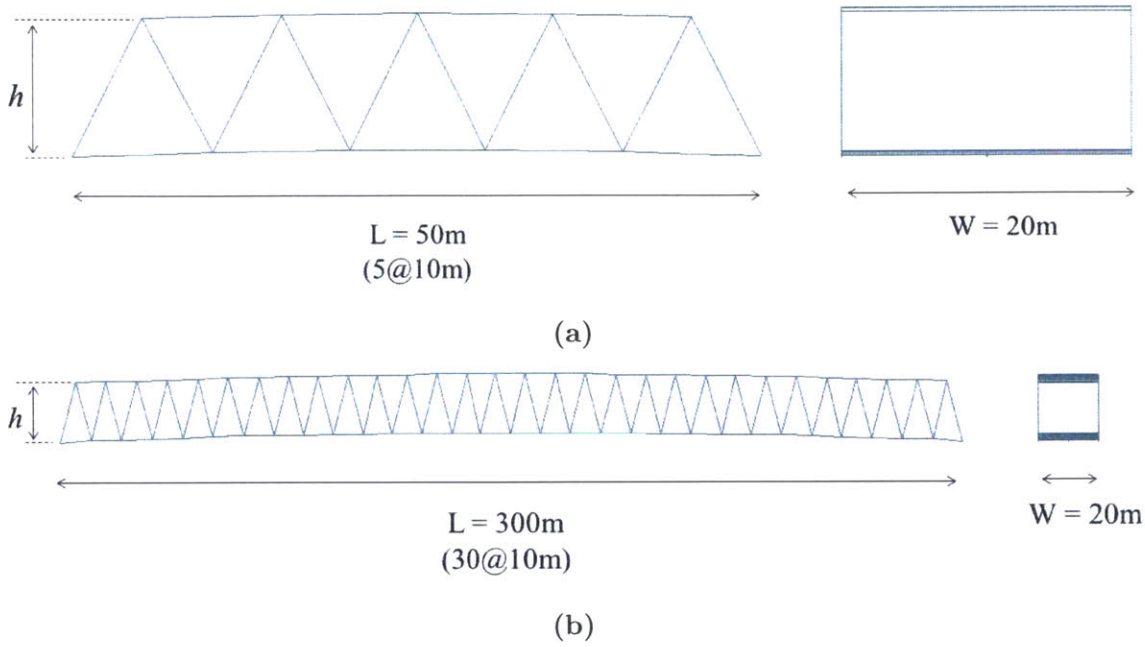


Figure B.6 – The additional Warren truss models span **(a)** 50 m and **(b)** 300 m. Both have the same parabolic variation in their top and bottom chords as the 200 m model shown in Chapter 3, and both have a single geometric sizing variable, h.

B.3 Experimental design: infeasible feature combinations

§3.1.3 described the experimental design, which specifies the set of optimization problems to be solved, as full factorial, indicating that we use all combinations of feature levels to generate problems. Strictly speaking, there are a few infeasible level combinations, and the design accordingly falls just short of full factorial. These infeasibilities stem from the following realities of structural design.

Each structural type is best suited to a limited range of spans. Not all structural types work well with the entire range of commonly-encountered bridge spans. Cable-stayed and suspension bridges, for example, are better suited to longer-span bridges, typically in excess of 300 m. The span of steel girder bridges, on the other hand, generally does not exceed 100 m or so. Table 3.3 shows the feasible combinations of the *structural type* and *span* factors used in the experimental design.

Mass cannot be constrained for mass minimization problems. "Mass" is a possible level of both the *objective type* and *constraint type* features. It does not make practical sense, however, to constrain mass when its minimization is already the objective of a problem.

Mass constraints cannot be omitted for stiffness optimization problems. The chosen formulation of the stiffness optimization problems (Eq. 3.2) asks algorithms to find the stiffest structure using a given amount of material. This upper-bound constraint on structural mass is necessary to prevent algorithms from simply adding large amounts of material in pursuit of a maximally stiff structure, and must therefore be present.

The geometry of girder bridges does not vary. The "combined geometry and sizing" level for the *problem type* factor does not make sense for girder bridge problems, since their geometry remains constant.

The stresses in cable-stayed bridges' towers must always be constrained. To ensure numerical stability of the structural models, we found it necessary to always constrain the stresses in the cable-stayed bridges' towers, in addition to constraining these bridges' deck displacements.

Appendix C

Additional results visualizations

This appendix contains additional visualizations of the relationships between design problem features and algorithm performance. §C.1 plots variations in incurred computational cost against the number of design variables in each problem and against structural type, and §C.2 shows the ranges of values behind the median-only plots used throughout Chapter 5.

C.1 Computational cost visualizations

We generally consider algorithms' solution quality to be of greater importance than their computational cost. Chapter 5, when exploring associations between problem features and algorithm performance, therefore deferred several of the computational cost plots to this appendix.

Fig. C.1 shows how each algorithm's performance varies with the number of design variables, complementing Fig. 5.15 in §5.3.3. Only CRS and ISRES display statistically significant ($p_s \leq 0.05$) associations; CRS makes fewer analysis calls as the problem size increases, and ISRES makes relatively more. The other six algorithms do not show strong trends. As with the computational cost plots in Chapter 5, local and global algorithms are normalized separately following Eq. 3.8.

Fig. C.2 is the computational cost version of Fig. 5.20 in §5.3.5. Only NEL-MEAD, PRAXIS, and CRS demonstrate substantial variation in their median computational cost (the ranges behind these medians are shown in Fig. C.8) across the six structural types; under this measure, the other five algorithms are less affected.

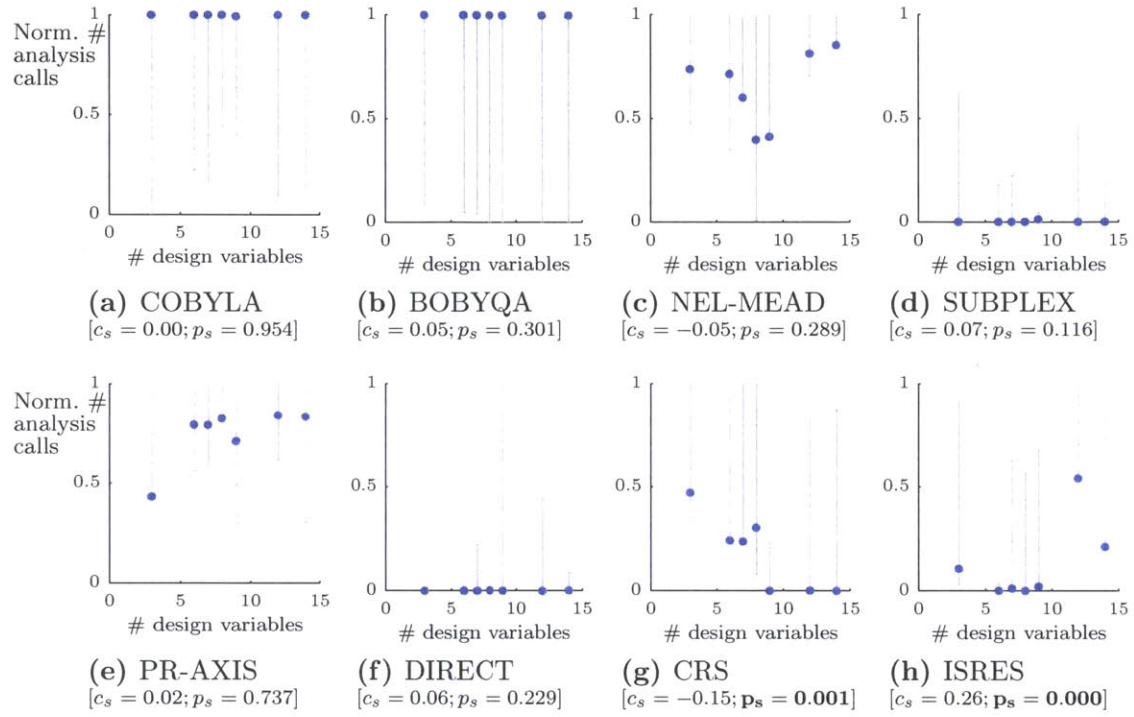
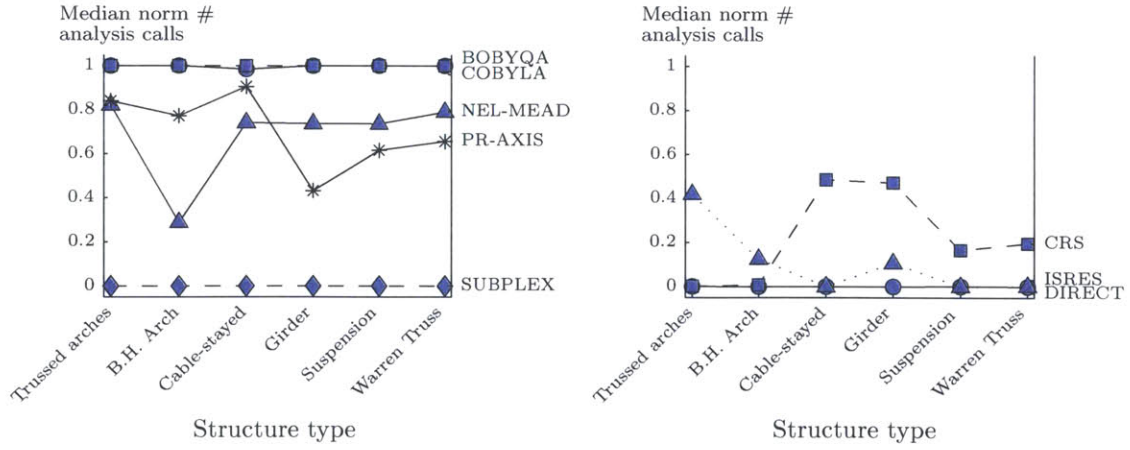


Figure C.1 – The median computational cost exhibits a statistically significant correlation with the number of design variables only for CRS and ISRES.



(a) Local algorithms **(b)** Global algorithms

Figure C.2 – Each algorithm's median normalized computational cost varies across the six structural types.

C.2 Performance ranges behind median values

Many of the data visualizations used to explore associations in §5.3 showed median values of algorithm performance across categories of problems. This section uses boxplots to show ranges of data behind these median values, providing additional information. Although these ranges do not explicitly appear in the Chapter 5, they are reflected in the Kruskal-Wallis coefficients used to test for statistically significant differences between groups.

A more complicated picture of algorithm performance across geometry and sizing problems emerges from Fig. C.3. Based on median values alone, we may conclude that ISRES should

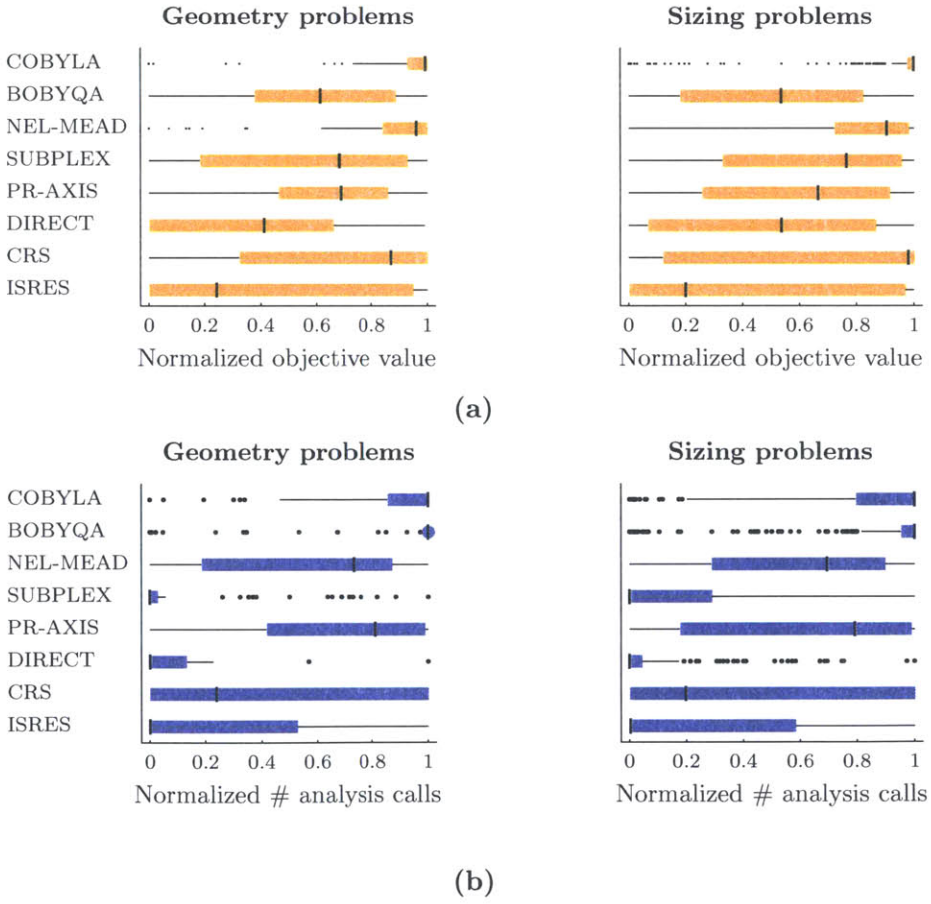


Figure C.3 – The range of data behind the median values shown in Figs. 5.11 and 5.12 present a more complicated picture of performance variation across geometry and sizing problems. Some algorithms, such as ISRES and DIRECT, which have relatively poor median objective values, often produce some of the best solutions, especially on sizing problems.

never be used to solve geometry problems. The additional information here, however, shows that ISRES often produces high-quality solutions to geometry problems and even, on occasion, finds the best solution.

Fig. C.4 shows algorithm performance ranges for problems with mass and stiffness as their objectives. Although CRS has a dramatically better median outcome on stiffness problems, its range of normalized objective values is widely spread. PR-AXIS has a worse median normalized objective value on stiffness problems, but its range—and the amount of relatively bad solutions it finds—is narrower than for mass problems, indicating less potential for downside.

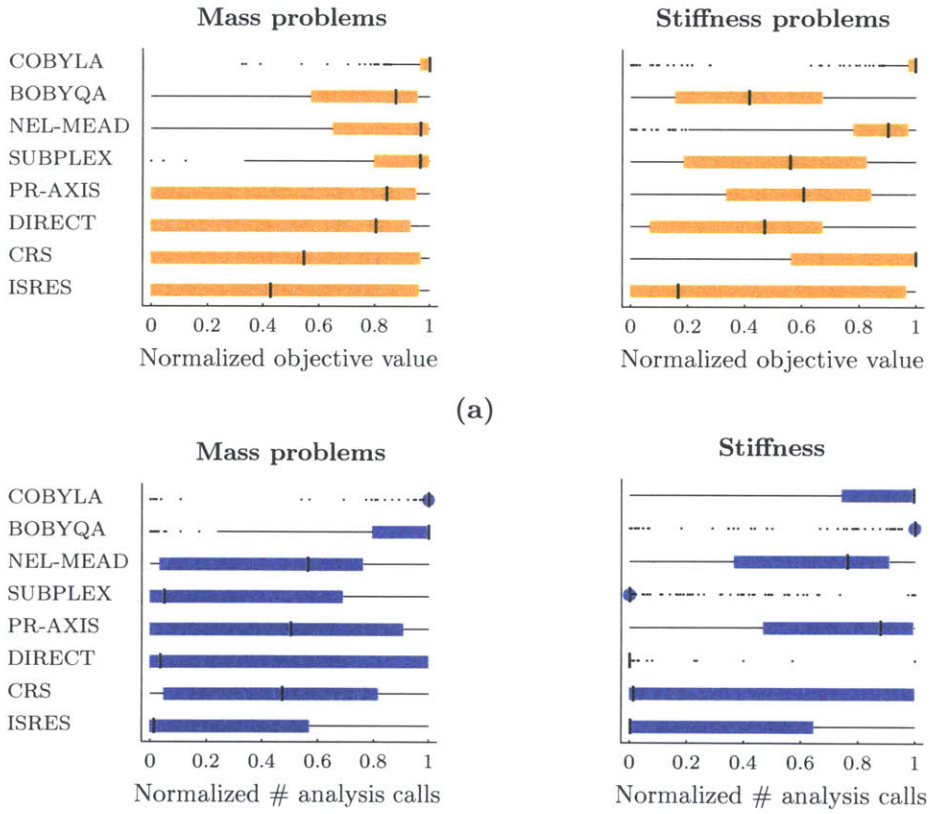


Figure C.4 – The ranges of data behind the median values shown in Figs. 5.13 and 5.14 show much more variation. Many algorithms, especially the global ones, exhibit a wide range of performance values around their medians.

Figs. C.7 and C.8 show the performance ranges when problems are split by structural type, and Figs. C.5 and C.6 show the same information when problems are split by the categories of constraint present. Both show additional variation beyond the median-only

plots in the main body of the chapter.

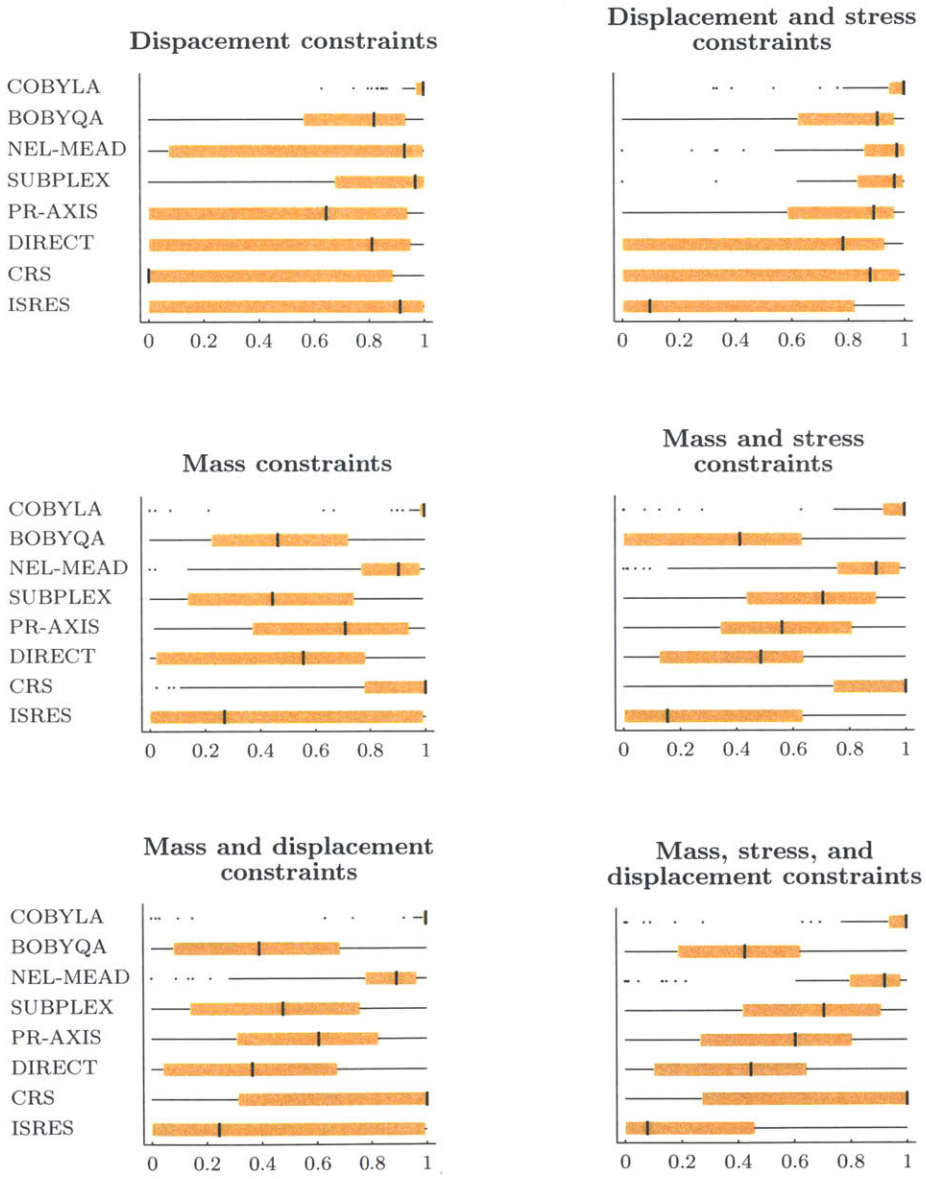


Figure C.5 – The observed range of solution quality achieved by each algorithm varies under the presence of various constraints, much more so than Fig. 5.18 indicates.

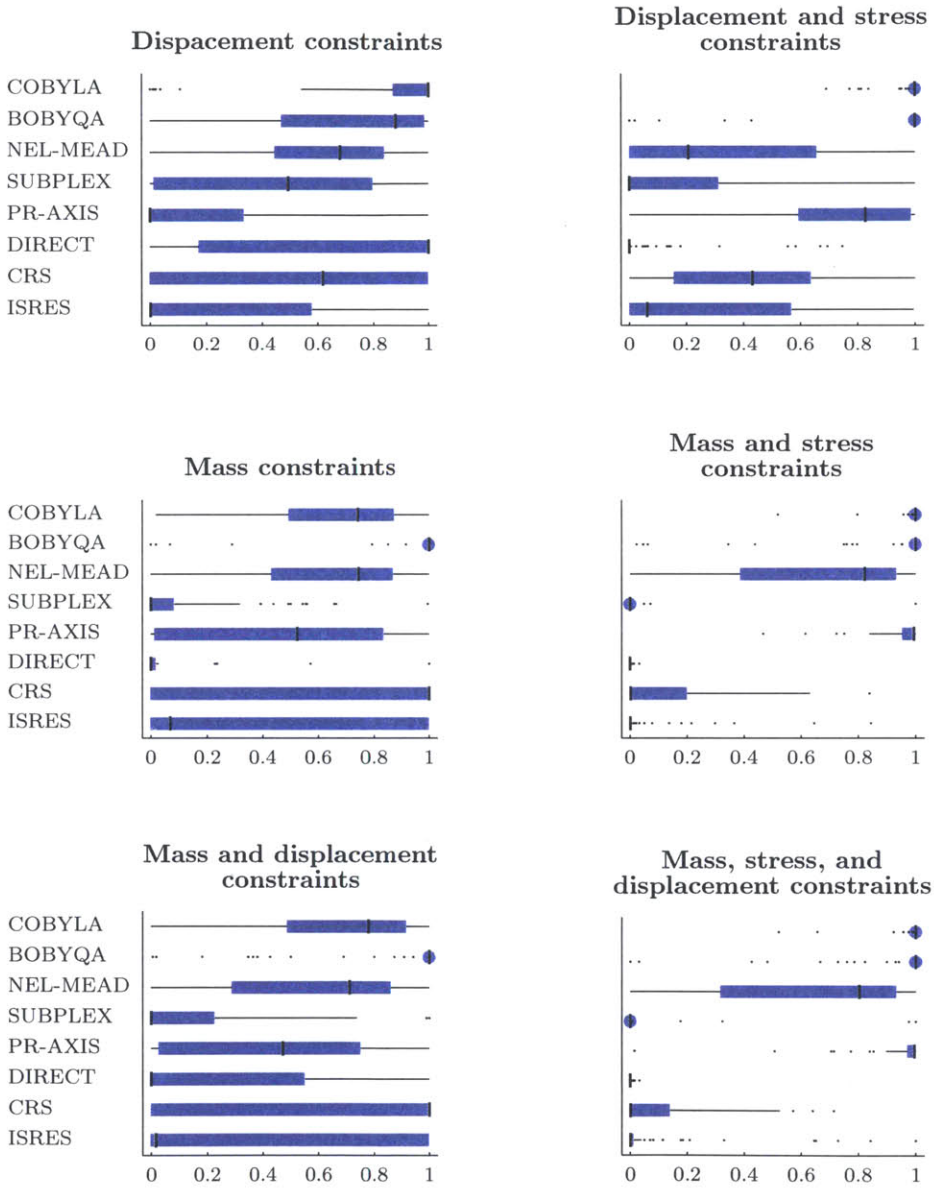


Figure C.6 – The observed range of computational cost incurred by each algorithm varies under the presence of various types of constraint. (Fig. 5.18 showed the median values only.)

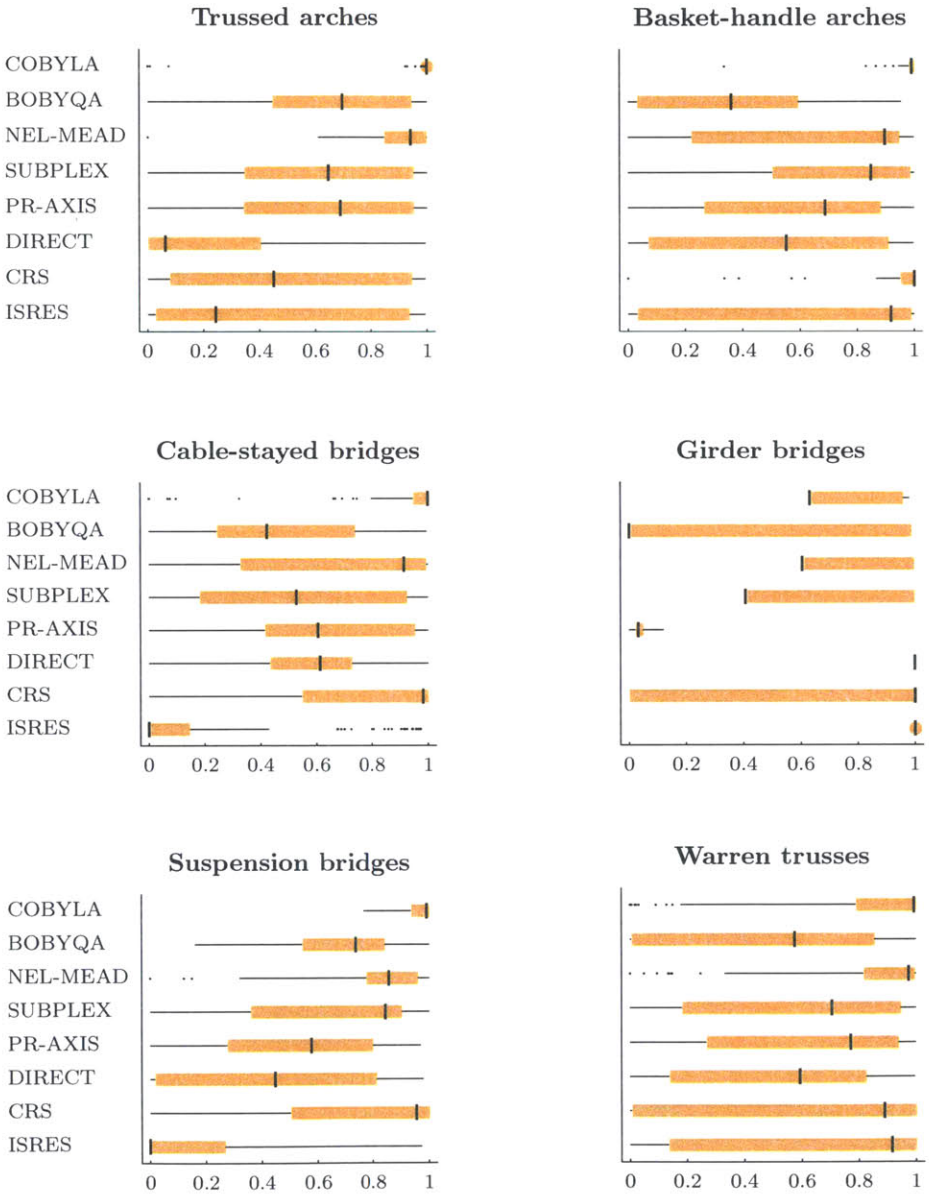


Figure C.7 – The range of data behind Fig. 5.20's median values shows additional variation in solution quality across the six structural types

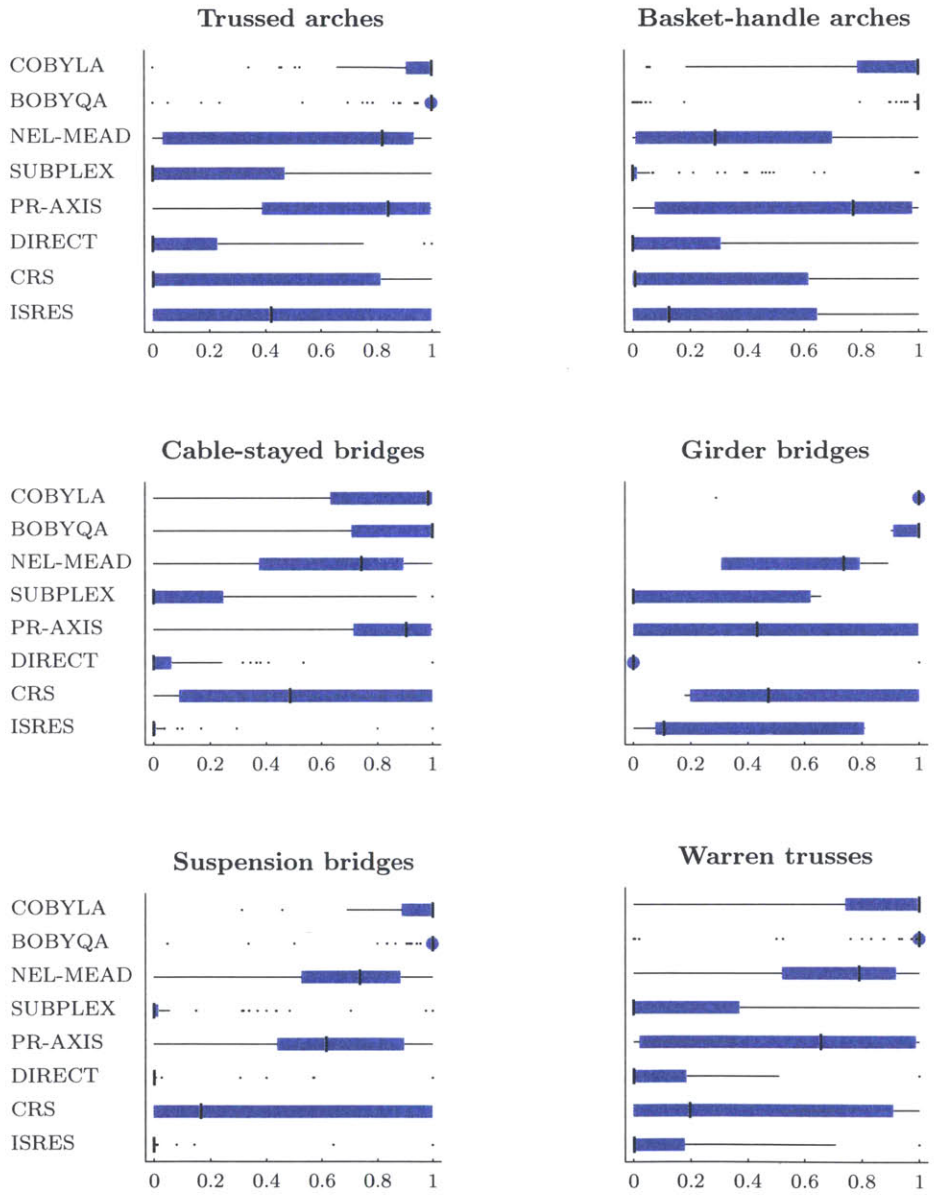


Figure C.8 – The range of data behind Fig. C.2's median values shows additional variation in computational cost across the six structural types

Appendix D

Automatically selecting the fastest algorithm

Chapter 6 presents computational techniques intended to select algorithms that produce the best solution for a given design problem. The same techniques—using the pattern classification and the regression-based approaches (§6.1)—are now used to select the algorithm which makes the fewest analysis software calls in converging on a solution, without regard to the quality of that solution.

Table D.1 shows how often each of the algorithm selection techniques chooses the computationally-lightest algorithm and how often each one chooses an algorithm whose number of analysis calls is within 10% of the lowest. This relaxed evaluation criterion still leads to a binary evaluation of success; the selected algorithm either lies within the specified percentage of the true best-performing algorithm or it does not. As in Chapter 6, the results are compared to two manual algorithm selection strategies: choosing a single algorithm (BOBYQA, in this case) for all problems and randomly choosing algorithms based on a uniform probability distribution.

All bar one of the developed techniques outperform the manual selection strategies on cross-validated test sets, by up to 11.2% on the strict evaluation criteria (where the relaxation factor $R = 0\%$) and by slightly less on the relaxed evaluation ($R = 10\%$), due to BOBYQA's proportionally greater improvement under the relaxed measure.

In future work, the two algorithm performance measures—solution quality and computational cost—could be weighted, and selection techniques could be used to choose an algorithm that satisfies a user-specified combination of high solution quality and low computational cost. In the early stages of design, for example, a selection technique could be adjusted to favor algorithms with lower computational cost, producing solutions quickly and allowing for quick iteration on design concepts. In later-stage design, an engineer

using this system could adjust the weighting in favor of algorithms that produce high-quality solutions. The additional computational expense that would likely accompany this shift would be less burdensome during late-stage design.

Table D.1 - Binary success rates for selecting the fastest algorithm

Algorithm selection technique	Cross-validated test data		Training data	
	R = 0%	R = 10%	R = 0%	R = 10%
Manual selection				
Manual - always BOBYQA	69.0	73.0	69.0	73.0
Manual - random choice	12.0	67.8	11.9	67.9
Pattern Classification formulation				
(a) Binary decision tree	76.1	77.6	81.7	82.7
(b) Boosted decision tree	80.2	81.0	97.4	98.1
(b) Random Forest	80.2	81.5	100.0	100.0
(c) K-nearest neighbor	77.4	78.7	83.4	84.4
(d) Artificial neural network	74.6	79.2	84.9	91.0
(e) Support vector classification	73.6	75.1	80.0	83.9
(f) Naïve Bayes	65.5	66.1	65.7	66.3
Regression-based formulation				
(a) Artificial neural network	74.7	75.5	78.0	83.8
(b) Support vector regression	76.8	78.7	80.2	82.3
(c) Regression Tree	78.2	81.1	79.3	85.2
(d) Generalized linear model	68.0	72.1	69.3	74.5
Best observed value	**80.2**	**81.5**	**100.0**	**100.0**
Improvement over manual selection	11.2	8.5	31.0	27.0