Optimization Constrained CAD Framework with ISO-Performing Design Generator

Kelly Eric Bowman
Brigham Young University - Provo

OPTIMIZATION CONSTRAINED CAD FRAMEWORK WITH ISO-PERFORMING DESIGN GENERATOR

by

K. Eric Bowman

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

December 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

K. Eric Bowman

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Christopher A. Mattson, Chair
Date	C. Greg Jensen
Date	Robert H. Todd

ABSTRACT

OPTIMIZATION CONSTRAINED CAD FRAMEWORK WITH

ISO-PERFORMING DESIGN GENERATOR

K. Eric Bowman

Department of Mechanical Engineering

Master of Science

Design decisions have a large impact early in the design process. Optimization methods can help engineers improve their early decision making, however, design problems are often ill-posed for optimization at this early stage. This Thesis develops engineering methods to use optimization during embodiment design, despite these difficulties.

One common difficulty in designing mechanical systems is in handling the effects that design changes in one subsystem have on another. This is made more difficult in early engineering design, when design information is preliminary. Increased efforts have been made to use numerical optimization methods in early engineering design – because of the large impact early decisions have on subsequent development activities. One step toward executing meaningful optimizations in early design is the development of an optimization framework to be used when conditions are expected to change as the design progresses and new information is gained. This Thesis presents a design framework that considers such change by subjecting the parametric updating of CAD models to optimization criteria specific to the problem at hand. Under the proposed framework, a part or subassembly is parametrically modeled in CAD; when changes are made to the subsystems that interact

with the part or subassembly, it is then updated subject to design objectives and constraints. In this way, the updated part or subassembly satisfies system and subsystem level optimization criteria, reducing the need for the designer to react to design changes manually. It is used to reduce the weight of a Formula SAE suspension rocker by 18%, demonstrating the utility of this framework. Next, we develop methods to help engineers by giving them options and helping them explore during configuration generation.

The design of multiple-bend, progressive-die-formed springs typically comprises four steps: (i) functional specification, (ii) configuration generation, (iii) configuration selection, and (iv) detailed shape and size optimization. Configuration generation fundamentally affects the success or failure of the design effort. This presents an important problem: by not generating potentially optimal configurations for further development in detailed design, the designer may unknowingly set the design on track for sub-optimal performance. In response, a method is developed that improves configuration generation. Specifically, an optimization-based spring configuration generator – without which, the generation would typically be based solely on designer creativity, experience, and knowledge. The proposed approach allows the designer to explore numerous optimization-generated spring configurations, which feasibly satisfy the functional specifications. The feasibility study is carried out *before* a final configuration is chosen for detailed development. Thus streamlining the designer's efforts to develop a design that avoids sub-optimality. We use the feasible-configuration generator to identify twenty-two electrical contact spring configurations. All twenty-two of the configurations satisfy the design's functional specifications.

Two important concepts that improve decision making in early design were chosen. First, is the concept of a paremetric CAD based framework. Second is the concept of generating iso-performing design solutions. A numerical computer-based application is explained that takes advantage of these two ideas. A genetic algorithm topology optimization framework with the ability to converge to iso-performing solutions was integrated with CATIA V5. This application is demonstrated on a Formula SAE frame where it develops a pareto frontier of designs, expands upon one compromise design by producing iso-performing solutions, and automatically produces designs with the same performance after a parametric suspension change.

ACKNOWLEDGMENTS

Dr. Mattson has helped me, so have my parents Dr. and Mrs. Bowman. We acknowledge ATL Technology and their engineers Marcus Darrington, Jaremy Flake, Matt Strong, Richard Liu, and Owen Carlson for inspiring the development of an optimization-based configuration generator for electronic contacts. Thank you to Dr. Todd and Dr. Jensen for being members of my committee. Thanks to Matthieu Pupat from Altair Engineering for the CAA/RADE tips. The authors thank the Fulton College of Engineering and Technology at Brigham Young University for funding the project. The authors also thank Mr. John Wakefield for his insightful comments.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Present and Future Optimization

Design decisions have a large impact early in the design process. Some of the most important decisions are those related to selecting design concepts or concept features. Many methods have been developed to help the designer in this important process of decision making [1, 2, 3, 4, 5, 6, 7]. These methods, however, are typically limited by the designer's ability to distinguish between a limited number of designs. A more rapid and comprehensive search method could be beneficial if it helps the designer find a more suitable design. Therefore, it is worth considering numerical optimization as a decision making tool for early design. Unfortunately, optimization at this early stage is difficult because relatively little is known about the problem at hand. The aim of this thesis is to propose and develop methods and tools to help engineers use the power of optimization methods to improve decision making earlier in the design process.

Five typical steps of the design process are (i) functional specification, (ii) concept generation, (iii) concept evaluation and selection (iv) embodiment design of the selected concept, and (v) detailed design. Usually, engineers first complete a feasible detailed design, and then use optimization methods to improve on that design as much as possible. Optimization could have a greater impact during early design, but there are a few factors that currently make this impractical. The goal of this thesis is to identify problems that keep engineers from using optimization in early design and solve them.

It is important to note at this point what we mean by early design. In Chapter 2 we explain that the earlier we are in the design process, the greater our decision impact can be. For this reason optimization can have the more impact the earlier it is used. At the same time, the earlier we are in the design process, the less knowledge we have, and the more problematic it is to use optimization. Currently, optimization is usually used in stage (v) mentioned above, detailed design. Optimization is can be very succesfully implemented at this stage already. ideally, we would want to use optimization in stage (ii), concept generation. However, not enough knowledge has been acquired by the conceptual design phase to use optimization unless much is already known from the beginning of the design project. The goal of this thesis is to develop methods that can be used in stage (iv), embodiment design (also called configuration generation, or conceptual design if the project is well defined). Throughout the rest of this thesis, let it be clear that we are proposing and developing methods to be used in this stage of design where there is a lot we do know and can apply, but we do not know everything and we expect our knowledge to change with time.

The first problem with using optimization earlier in design is loss of work due to changing design requirements. It is very common for an engineer to spend time perfecting the design of a particular component, only to see his work wasted because an interfacing subsystem changes in a way that forces his component to change along with it. Changes like this prevent designers from taking the time to set up an optimization problem because they are aware that when a change is required, most work invested will be lost. It would be useful to have a tool that prevents the loss of work that has been invested in an optimization formulation. The second problem with using optimization earlier in design is that many aspects of the design are still unknown and unmodeled. As a result, it is important to present the engineer with design options and help him/her explore the design space instead of considering a single design that may not capture important objectives and constraints.

1.2 Thesis Structure

In this, the first Chapter, we outline the structure of this thesis. Chapters 2 and 3 are published papers that deal with separate topics relating to the greater problem: using optimization to improve decision making during embodiment design. Chapter 4 is an integrated computer application that takes advantage of principles described in the two published papers. The final Chapter summarizes the findings from Chapters 2 through 4. The following sections provide more detail regarding the content of the remaining Chapters.

1.3 Theoretical Optimization Constrained Framework

In the second Chapter, we present "Handling Frequent Design Changes by Automatic Optimization-Constrained Updates of Parametric CAD Models" [8]. This Chapter is critical because it introduces a theoretical framework that is used in Chapter 4 to integrate numerical optimization with parametric CAD. A brief overview is provided. A common difficulty in designing mechanical systems is in handling the effects that design changes in one subsystem have on another, or on the system as a whole. This is made more difficult in early engineering design, when frequent changes are required and design information is preliminary. Increased efforts have been made to capitalize on the benefits of numerical optimization methods (search methods) in early engineering design – because of the large impact early decisions have on subsequent development activities. An important step toward executing meaningful optimizations in the early stages of design is the development of a design optimization framework that can be used when objectives, constraints, variables, and other conditions are expected to change as the design progresses and new information is gained. Chapter 2 presents a design framework that considers such change by subjecting the parametric updating of CAD models to optimization criteria. Under the framework proposed in Chapter 2, a part is generically and parametrically modeled in a CAD system; when changes are made to the design of subsystems that interact with the part, the part is then automatically updated subject to design objectives and constraints. In this way, the updated part or subassembly satisfies system and subsystem level optimization criteria. Thus reducing the need for the designer to react to design changes in one subsystem by manually correcting the affected design of another. The framework proposed in Chapter 2 carries practical implications that are demonstrated in the development of a suspension rocker for a formula SAE car designed and built at Brigham Young University, resulting in a rocker weight savings of 18%.

1.4 Feasible Configuration Generator

In the third Chapter we present “Feasible-Concept Generator for Multiple-Bend Springs” [9]. The feasible configuration generator is critical because it brings the notion of iso-performance to the Thesis albeit in the context of multiple bend springs used for electrical connector systems. A brief overview is provided. The design of multiple-bend, progressive-die-formed springs typically comprises four iterative steps: (i) functional specification, (ii) configuration generation, (iii) configuration evaluation and selection, and (iv) detailed shape and size optimization of the selected configuration. The process of configuration generation fundamentally affects the success or failure of the design effort. Specifically, by not generating potentially optimal configurations for further development in detailed design, the designer may unknowingly set the design on track for sub-optimal performance by selecting a configuration that does not have the potential of some other configuration. In response, the approach presented in Chapter 3 focuses on improving the important process of configuration generation. Specifically, Chapter 3 presents an optimization-based spring configuration generator – without which, the generation would typically be based solely on designer creativity, experience, and knowledge. The approach in Chapter 3 allows the designer to explore numerous optimization-generated spring configurations, which are generated specifically to feasibly satisfy his or her functional specifications. Importantly, the feasibility study is carried out *before* a final configuration is chosen for detailed development. Thus streamlining the designer’s efforts to develop a design that has a high possibility of avoiding sub-optimality. As shown in Chapter 3, we use the feasible-configuration generator to identify twenty-two specific spring configurations for the design of an electrical contact. All twenty-two of the configurations satisfy the design’s functional specifications.

1.5 Practical Optimization Constrained Framework With Iso-Performance

In the fourth Chapter, an integrated computer application is presented that brings the theoretical framework described in Chapter 2 into practice. It accomplishes this by creating CATIA V5 commands. These commands allow a designer to create a geometrical model, a structural analysis model and an optimization model at the same time. This integrated model can be parametrically linked to other CAD geometry, and when any parametric change is made to the CAD model, the user can update the the integrated model without repeating any previous work. In addition, it extends the method described in Chapter 3 beyond the design of electrical contact springs to the design of any structure that can be modeled with beam elements.

1.6 Chapter Summary

In this Chapter we outlined the structure of the Thesis. The following Chapter is a paper published in the ASME International Mechanical Engineering Conference and Exposition in 2007, the third Chapter is a paper published at the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference in 2006. Following these, a CATIA V5 application that performs optimization constrained updates as described in Chapter 2 and can produce iso-performing designs as described in Chapter 3 is presented. Finally a concluding Chapter presents an overall summary of conclusions as well as a list of possible future work.

Chapter 2

Handling Frequent Design Changes by Automatic Optimization-Constrained Updates of Parametric CAD Models

2.1 Description

This paper was published at the ASME International Mechanical Engineering Conference and Exposition in 2007 by K. Eric Bowman and Christopher A. Mattson. The full reference for the paper is: Bowman, K. E., and Mattson, C. A., 2007. Handling frequent design changes by automatic optimization-constrained updates of parametric CAD models Tech. Rep. IMECE2007-42379, ASME International Mechanical Engineering Conference and Exposition, November.

2.2 Introduction and Literature Survey

Throughout the development process new information about the design-at-hand is continually becoming available. It is the designer's responsibility to seek out and understand this information and to use it to make design decisions that bring the development efforts to better and better designs. The development of mechanical systems carries a particular challenge when it comes to new information that arises from design change and iteration; the challenge of handling the effects that design changes in one subsystem have on another, or on the system as a whole. For parts and subsystems that are not easily adaptable physically, or not designed in a flexible way, the interdependence of parts often results in change requests that are unwelcome and discouraged among subteams – ultimately slowing the development progress. These problems are more difficult to handle in early engineering design, when frequent changes are made and design information is preliminary.

This paper presents a design optimization framework that expects frequent changes in design objectives, constraints, variables, and other conditions that naturally arise as the design progresses. Under the proposed framework, a part (or component) is generically and parametrically modeled in a CAD system; when changes are made to the design of subsystems that interact with the part, the part is then automatically updated subject to design objectives and constraints. This requires the parametric updating of CAD models to happen according to the design optimization criteria related to the design at hand.

One of the main benefits of the proposed method is that it reduces the designer's need to *react* to the design changes in one subsystem by manually correcting the affected design of another. The optimization-based framework presented herein automatically updates the parametric models subject to conditions that maintain their design functionality and optimality. Thereby allowing the designer to monitor the optimization-driven changes, as opposed to making the changes him/herself. The following subsection reviews the literature in the areas of (i) optimization methods for early design, and (ii) methods for dealing with uncertain information in early design.

2.2.1 Literature Survey

It is natural that a product in the detailed design phase has a more stable, more developed mathematical description than a product in the conceptual (early) design phase. As such it is logical to use numerical optimization during detailed design when mathematical models are stable and well posed. It is during detailed design that engineers create detailed analytical models that describe the performance of a system, subsystem, or part. These analytical models can then be used as objective functions or constraints in an optimization problem statement. Using traditional optimization methods during conceptual design is more difficult because most designs are not well posed at this early stage. However, it would be beneficial to capitalize on the power of numerical search methods as early as possible because the conceptual design phase has a large impact on all later phases and because optimization-based search methods are known to effectively characterize the design space. Also, it is well-known that the final design is limited by the concept chosen at this early, uncertain, stage. In short, we seek to select a concept in the most effective way possible.

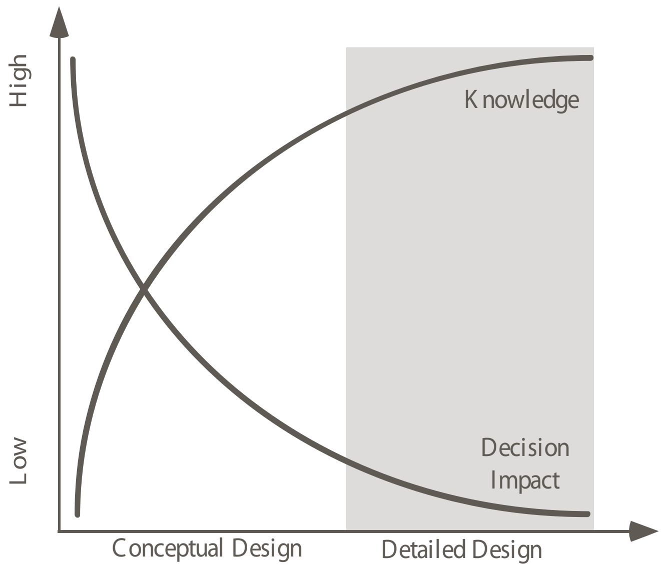


Figure 2.1: Decisions with the most weight are made when knowledge is the least.

Many have indicated that most of the product's success-dependent characteristics are decided in the early design phases [10]. Figure 2.2.1 illustrates this phenomena. The horizontal axis represents time, with two major divisions noted as *conceptual design* and *detailed design*. The vertical axis is an increasing scale from *low* to *high*. The two curves represent increasing design knowledge and decreasing impact of decisions as time increases.

Because of the important role that optimization methods can have on the success of a product design, there have been increased efforts to use optimization during conceptual design. Two common approaches emerge in the literature. The first approach is to generalize a design. An example is Morino's work with aircraft design [1]. Aircraft design is a well developed field. Because it is well developed, engineers have a good feel for what will and will not work early in the design process. They also know what kinds of analysis will be necessary from the beginning. As a result of this early knowledge, it is possible to create a generic airplane model that can describe numerous feasible aircraft configurations. The parameters for a specific airplane design can then be input into this model and the configuration can be optimized. This same approach has been succesfully used in other well developed fields [2, 3].

The second approach is topology optimization [4, 5, 6, 7]. In topology optimization, a design domain is descritized and each element represents a design variable. Depending on the type of analysis being run, the design variable could represent element existence, element density, or some other element property. This method essentially searches through all designs that could fit within the design domain. In this way, the model is specific to an application, and for an entire analytical discipline. This method is flexible and can often be effectively used during conceptual design as long as the design domain can be descritized according to a finite element or finite difference model and the boundary conditions are known. Sometimes, however, even more flexibility than these methods allow is necessary.

The two approaches discussed above, require that the designers already have a high level of knowledge about the product. During conceptual design, when critical design decisions are being made, there is often a lack of design knowledge. Usually, when this is the case, optimization is deemed by most to be impossible. Instead of using optimization, designers have traditionally used all information available along with their experience and engineering intuition to make decisions. While designer knowledge and experience is not to be discredited, the use of optimization during conceptual design can help decision making in various ways. Namely, it can identify the best design, given the information available; it can help designers visualize a design space; it can also help eliminate feasible designs that are not optimal, thus simplifying the decision-making process; it can be used early to set up component design that can be automated, thereby reducing the need for the designers to do time-consuming detailed design of certain parts.

An important step toward executing meaningful optimizations in early design is the development of a design optimization framework that can be used when objectives, constraints, variables, and other conditions are expected to change as the design progresses and new information is gained. Here, we make the important connection that since future design changes are unknown they are a significant source of uncertainty in earlier phases of design. Therefore, in the following paragraphs we examine the literature on the topic of handling uncertainty in design optimization.

One approach is Optimization Under Uncertainty (OUU) [11]. The most common type of uncertainty considered in OUU is stochastic uncertainty or irreducible uncertainty [11]. This is commonly seen as a probability distribution about a mean. Stochastic uncertainty is not treated in this paper because it is present in all parts of design. Instead, this paper deals with uncertainty that is significantly present in conceptual design; epistemic uncertainty or reducible uncertainty [12, 13]. Epistemic uncertainty refers to uncertainty that comes from the lack of knowledge. One approach for handling epistemic uncertainty is evidence theory or Dempster-Shafer theory. In this theory, upper and lower bounds to probabilistic uncertainty are created. As more knowledge is acquired these bounds become the probabilistic distribution. These treatments of epistemic uncertainty deal with it in cases where it can be reduced to a probabilistic distribution.

While the present paper does deal with uncertainty that comes from lack of knowledge, it does not attempt to reduce it to a probabilistic distribution. This paper assumes that although the level of knowledge may not be complete in early design, as the design process progresses any necessary knowledge will become available. Here, optimization can be used with all available information to make the best decision possible *at any given moment*. If the optimization problem statement and framework under which it is optimized is able to accept changes later, then as more information is gathered, the decisions made early in the process can be improved upon. This requires an optimization framework with a notable degree of flexibility.

The need for flexibility is considered from two important perspectives; (i) from the perspective that in early design, little is known and therefore much is subject to change; and (ii) from the perspective that when parts of a system change interacting parts will consequently need to adapt and change quickly. Both are discussed briefly.

It is not uncommon to enter the conceptual design phase with only a partial understanding of a product's objectives and constraints. If an optimum is chosen early in the design process without considering the possibility of future changes to the optimization statement, then when new information is added, the optimum chosen previously is likely no longer optimal. For example, the solution to a single objective optimization problem is a single point. If, later on, another objective is added, the optimum is no longer a single point, but rather a Pareto set. This Pareto set can be represented in objective space as a Pareto frontier. The optimal point from the single-objective optimization is an endpoint, or anchor point, of the Pareto frontier. While this point is Pareto optimal, it is only one of many optimal designs and it is less likely that it would be chosen over a compromise design.

As mentioned previously, the design of mechanical systems can be particularly challenging as changes to one subsystem can often have adverse effects on other subsystems. The effects can often require fundamental changes to the definition of other subsystems. At its most fundamental level, the changes in one subsystem can result in system or subsystem changes in design objectives, design constraints, and design variables considered.

This paper presents a design framework that is flexible and can accommodate the kinds of changes discussed above. The framework is CAD and optimization based. As will be shown, parametric CAD models will be updated subject to optimization conditions that ensure that the update is favorable and that it satisfies the conditions of the design problem.

The remainder of the paper is organized as follows. In Section 2.3 we explore the fundamental changes that an optimization problem statement may undergo as design information increases. Section 2.4 presents the framework for automatic optimization-constrained updating of parametric CAD models. The design of a suspension rocker arm for BYU's formula SAE car is presented in Section 2.5, followed by concluding remarks in Section 2.6.

2.3 Fundamental Changes To Optimization Problems as Design Information Increases

One important step toward using optimization in an environment with changing information is to understand the different ways in which an optimization problem statement can change as the product progresses through its development. In this section the different parts of an optimization statement are analyzed, and different ways in which the optimization statement can change are illustrated. We begin by examining a generic optimization problem statement (Sec. 2.3.1). We then consider changes to the design objective (Sec. 2.3.2), the design constraint (Sec. 2.3.3), and the design variables (Sec. 2.3.4), respectively.

2.3.1 Generic Optimization Problem Statement

A general optimization statement is as follows:

$$\min_{x}\{\mu_1(x), \mu_2(x), ...\mu_n(x)\} \tag{2.1}$$

subject to:

$$g_q(x) \le 0 \quad q = 1, ..., r \tag{2.2}$$

$$h_j(x) = 0 \quad j = 1, ..., v \tag{2.3}$$

$$x_{il} \le x_i \le x_{iu} \quad i = 1, ..., n_x \tag{2.4}$$

where $\mu_i(x)$ represents the ith objective, $g_q(x)$ and $h_j(x)$ represent the constraints, and all x_i represent the design variables. When this generic optimization problem is made specific, there will be n objectives, r inequality constraints, v equality constraints and n_x variables – at the time of establishing the problem statement. As the design progresses, the objectives, constraints, and variables can be modified in three ways. An objective, constraint or design variable can be added, removed or changed. Since the effects of removing an objective or constraint can be seen as the reverse of adding one, it is only necessary to observe the effects of adding an objective, constraint or design variable. This creates six basic ways in which an optimization problem can be modified. They are: an objective can be added, an objective can be changed, a constraint can be added, a constraint can be changed, a design variable can be added and a design variable can be changed. By studying these six basic changes that can occur, more complex changes can be understood. The following subsections explore these different ways in which an optimization statement can change.

2.3.2 Added or Changed Design Objectives

Because of the inherent lack of knowledge during conceptual design, there are many reasons an objective might be changed later in the design process. If optimization is used during conceptual design, it is important to know how the product will be effected by a changed objective. If an objective function is changed, then the result is a potential change

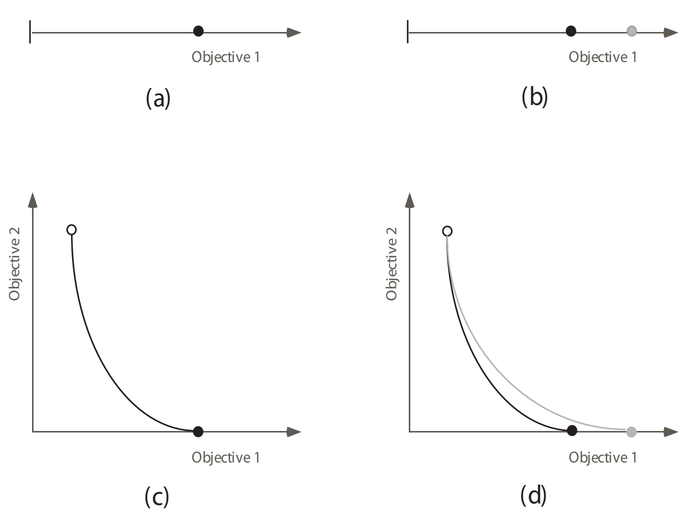


Figure 2.2: The effects of changes in an objective viewed in objective space. (a) One objective, (b) One objective that has been changed, (c) One objective added, and (d) One objective added and one objective changed.

in not only the design values that achieve the optimum result, but also in the value of the optimum result. In the design objective space this would appear as a shifting optimum. Figures 2.2(a) and 2.2(b) represent a single objective design space. As shown in Fig. 2.2(a) the optimum of a single objective optimization problem is represented as a black dot. Figure 2.2(b) shows the optimum from Fig. 2.2(a) as a black dot and a shifted optimum that could result from a changed objective as a grey dot.

In the same way that objectives can be expected to change due to the inherent uncertainty early in design, objectives can also be added or removed. In single-objective optimization the optimum set consists of a single point. If an objective is added, the optimum set changes from a single point to many points that are called the Pareto set. These points are often plotted in objective space as a Pareto frontier. The optimum point from a single-objective optimization problem becomes the anchor point of the new Pareto set. In Fig. 2.2(c) we see the bi-objective design space defined by Objective 1 and Objective 2. The figure shows the optimum (black dot) from Fig. 2.2(a) along with an open dot and a curve connecting the two. The open dot represents the anchor point for the other objective and the curve represents the Pareto frontier or the set of optimal (non-dominated) designs. If an objective is changed and an objective is added, the result appears as a shift in the Pareto frontier in objective space. In Fig. 2.2(d) the combined effects of changing an objective and adding an objective can be seen.

2.3.3 Added or Changed Design Constraints

The circumstances under which a constraint may change are very similar to the circumstances under which an objective may change. The effect of changing a constraint in the optimization problem is simply a change in the feasible design space as that constraint moves. The feasible design space is the set of all designs that satisfy all constraints. In Fig. 2.3(a) the dark gray portion of the oval represents the feasible design space. Just as changing an objective appears in objective space as a shifting optimum, changing a constraint appears in objective space as a shifting constraint. The black curve on the right-hand side of the figure represents a constraint. The light gray portions of the oval represent areas that are not feasible because of the constraints. In Fig. 2.3(b) the constraint has been changed, and therefore moved in the design space, and as a result the light gray region that represents non-feasible designs is larger.

If a new constraint is added, the effect is the addition of a new boundary in objective space. In Fig. 2.3(c) a new black curve appears in the left-hand side of the figure. The curve eliminates a new area of the feasible design space, which can be seen in light gray. This boundary may or may not effect the optimum design. If a constraint is changed and a constraint is added, the effect in objective space is a shift in the changed constraint and the appearance of a new constraint boundary. Figure 2.3(d) shows the combined effect of changing and adding a constraint.

2.3.4 Added or Changed Design Variables

The result of changing design variables can be observed in design variable space. In Fig. 2.4(a) shows the design variable space for a single variable case; the horizontal axis represents the design variable value and the vertical axis represents the objective function value. Changing the design variables could result in different objective values for given design variable values. This appears as a different curve in Fig. 2.4(b).

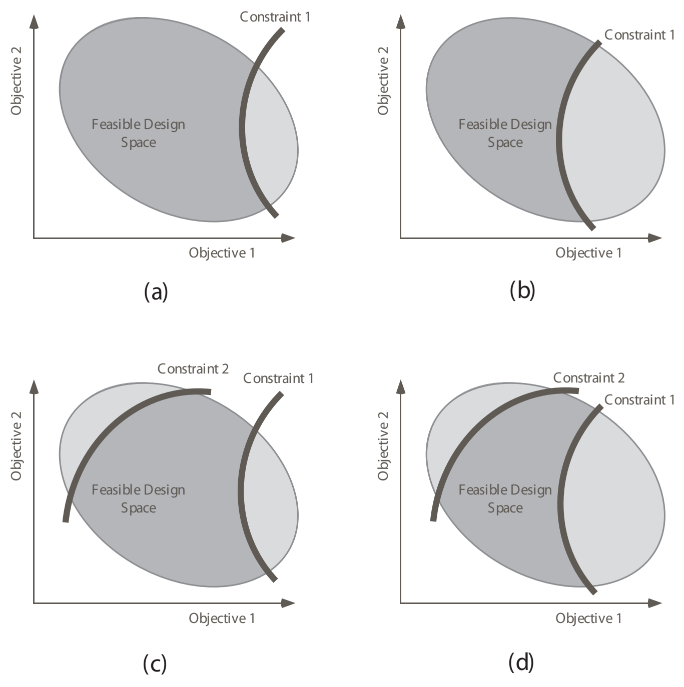


Figure 2.3: The effects of changes in a constraint viewed in objective space. (a) One constraint, (b) One constraint that has been changed, (c) One constraint added, and (d) One constraint added and one constraint changed.

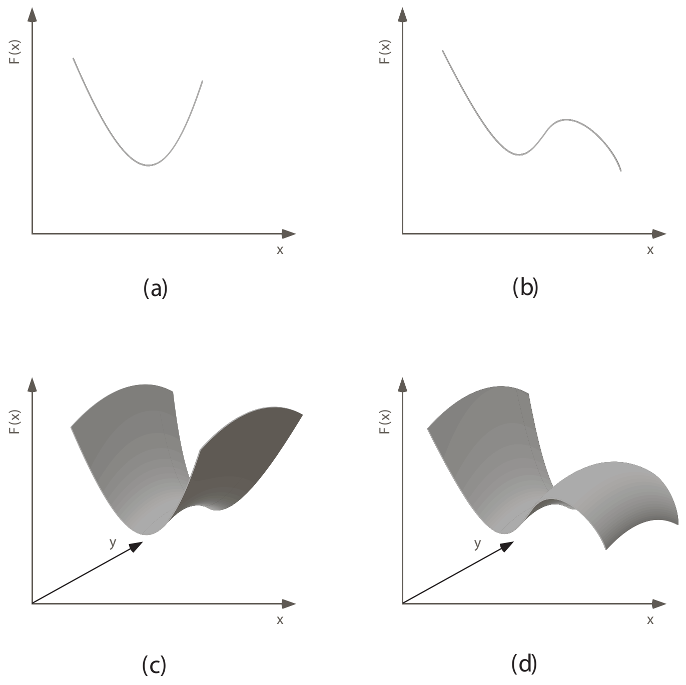


Figure 2.4: The effects of changes in the design variables as viewed in design variable space. (a) One variable, (b) One variable that has been changed, (c) One variable added, and (d) One variable added and one variable changed.

If a variable is added the result is the expansion of the design variable space into another dimension as can be seen by the surface in Fig. 2.4(c). Figure 2.4(d) shows the surface that results from adding a design variable and changing one.

2.3.5 A Comment on the Changing Nature of the Optimization Problem

As objectives, constraints, and design variables are added and changed, it may appear that nothing seems to stay the same in an optimization problem. As such one may conclude that if nothing remains from one change to the next, then early optimization information is not useful. It is true that if early information is completely incorrect, then an optimization problem would change completely as the project continued and in this case, the early optimizations would indeed be useless. However, it is rarely the case, if ever, that the information available early in the design is completely incorrect. Rather it is typically somewhat correct. Our experience is that if an objective is added, it was usually suspected early on that it may have to be considered. If it is changed, it is usually a small change, resulting in an optimum close to the original one. The same applies to added or changed constraints or design variables. This practical reality can be taken advantage of to create a flexible optimization problem that can handle the most probable changes in a given situation. The section that follows presents such a flexible framework.

2.4 A Framework for Optimization-Constrained Updating of Parametric CAD Models

If an optimization problem is created in such a way that it can accept and adapt to change, then it could be used during conceptual design and be a valuable tool to help engineers make the best decision possible amongst uncertain or changing information. In this section we develop an optimization framework that can accomplish this when tied to a parametric CAD assembly.

2.4.1 Optimization Applied to Parametric CAD/CAE

For the designer using optimization during early design, it would be problematic to assume that the obtained optimum would still be optimal at the end of the product development process. If, on the other hand, the optimization process is viewed as one that is continuously carried out over the product development process, then it is natural to think of the optima obtained in early design as a helpful step toward a final optimal design. We may even consider the optima obtained in early design to be *momentary optima* and where the final optimal solution is found by passing through many momentary optima as the development process continues.

The notion of *continuously carried out optimization* can be accomplished by coupling two of the most widely used tools in design; Parametric CAD Modeling, and Numerical Optimization. This coupling can be accomplished by representing the part to be optimized as a parametric CAD part whose parameters are variables in an optimization problem. The optimization problem receives inputs from the CAD assembly, processes them, and outputs objective values back into the assembly that are interpreted as the new part. In this manner, if the CAD assembly is changed, the optimization is simply re-evaluated. If something inside the optimization problem is changed, the algorithm is re-evaluated and the assembly simply receives the new output and adjusts accordingly. As long as the information interface between the CAD assembly and the optimization problem is robust, the model will be able to accept large changes without failing. Defining interfaces in this manner is a common practice in parametric CAD modeling of assemblies.

In a parametric CAD model, all important dimensions of a component or assembly are variable parameters instead of fixed values. In this way, if one component must be changed, linked components can adapt automatically instead of having to be remade from scratch. The mathematical relationships in an optimization problem are more complex and require more time to evaluate than the mathematical relationships in a parametric CAD model, yet they can be formed in a similar way.

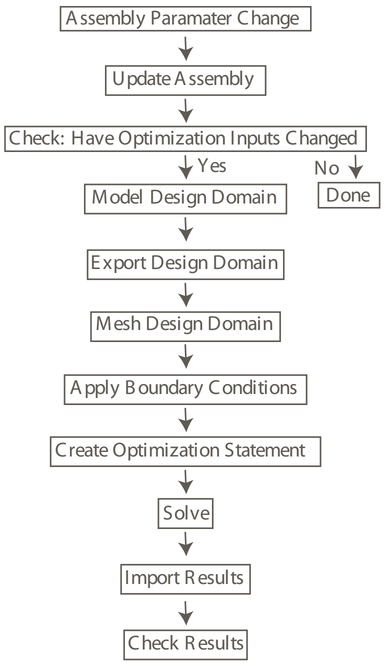

Assembly Paramater Change
Update Assembly
Check: Have Optimization Inputs Changed
Yes
No
Model Design Domain
Done
Export Design Domain
Mesh Design Domain
Apply Boundary Conditions
Create Optimization Statement
Solve
Import Results
Check Results

Figure 2.5: The parametric optimization process

2.4.2 Flexible Structural Topology Optimization

One way this parametric implementation could be used would be in the topology optimization of a structure. The problem optimization statement for the flexible topology optimization of a structure is as follows:

$$\min_x J = \sum_{i=1}^{n} x_i v_i \tag{2.5}$$

subject to:

$$\delta_j \le \delta_a \quad j = 1, ..., m \tag{2.6}$$

$$\sigma_i \le \sigma_y \quad i = 1, ..., n \tag{2.7}$$

$$\Omega = \bigcup_{i=1}^{n} \Omega_i \quad i = 1, ..., n \tag{2.8}$$

$$\Omega_i \cap \Omega_j = \emptyset, i \neq j \quad i, j = i, ..., n \tag{2.9}$$

$$0 \le x_i \le 1 \quad i = 1, ..., n_x \tag{2.10}$$

$$x_I = 1 \tag{2.11}$$

where J is the mass found by summing the product of the density, x_i, and the volume ,v_i, of each element, Ω_i, in the design domain, Ω. There are n elements in the design domain. δ_j represents the displacement of m nodes of interest, which must be less than some acceptable displacement, δ_a. σ_i represents the stress of each element. x_I is the density of any element Ω_I that lies on any interface between the part being optimized and the system it is in. The subscript I indicates that it is at the interface.

This optimization can be carried out according to the flow diagram shown in Fig. 2.5. First a parametric CAD model must be created that can provide the necessary inputs to the optimization problem, namely: the design domain, the boundary conditions, the objectives, and the constraints. The design domain for this kind of a problem could be a CAD model that represents all necessary interfaces between the structure and its surrounding system as well as the space available for it to occupy. The load case could be input from an outside source; if a change in the assembly has an effect on the load conditions, a method for automatically updating the load case would be necessary. Once the assembly has been updated a check is done to see if the update has effected the part in question. If it has, then the design domain and load case should be exported to meshing software where the design domain is meshed, and the load case is applied. Once the load case has been applied, the objectives and constraints are associated with the model and the topology optimization is run. The optimal design can then be returned to the outside environment along with any necessary output values. An engineer can then check to make sure that the optimization results are acceptable. If they are acceptable than the process is done.

2.5 Conceptual Design-Optimization of a Formula SAE Suspension Rocker

Flexible structural topology optimization can be applied to the design of a Formula SAE suspension rocker. First, Formula SAE will be described along with the design problem of suspension rockers. Second, the actual implementation of this method will be described.

Figure 2.6: The BYU FSAE Car

2.5.1 Formula SAE

Formula SAE (FSAE) is a student competition hosted by the Society of Automotive Engineers (SAE). In this competition, student engineers build formula-style race cars with limits on the car frame and engine. A student designed and built FSAE car is shown in Fig. 2.6. The suspension types of choice for the Brigham Young University FSAE team are pushrod and pullrod suspension. One important component of both pushrod and pullrod suspension is the rocker (sometimes called the bellcrank). By decreasing the weight of this part, not only the total weight of the car will be decreased but also the unsprung weight of the suspension. An example of pushrod suspension with a rocker can be seen in Fig. 2.7.

The purpose of the rocker is to give the suspension the correct motion ratio. The motion ratio is a the ratio of how far the suspension articulates at the wheel to how far the damper and spring move. The motion ratio is a critical parameter in suspension design and if it is not correct, the vehicle will not perform as predicted. There are a few ways in which the rocker design effects the motion ratio. Fig. 2.8 shows that the most obvious parameters that effect motion ratio are the lengths L1 and L2 of the rocker. The rocker concept is used because it gives the designer the ability to choose these two lengths and thus choose the best motion ratio. There are, however, other parameters that can adversely effect motion ratio. These are the angle between the shock and the rocker at ride height and the angle between the push-rod and the rocker at ride height. Ride height is the position

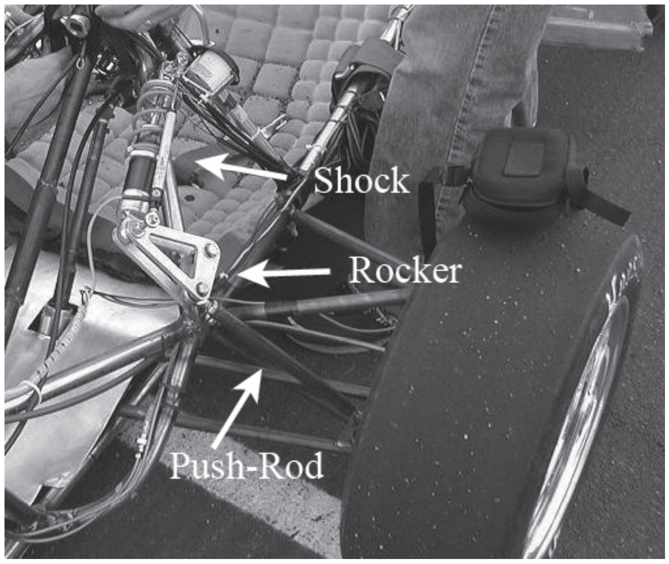


Figure 2.7: Pushrod suspension. The rocker can be seen in the picture.

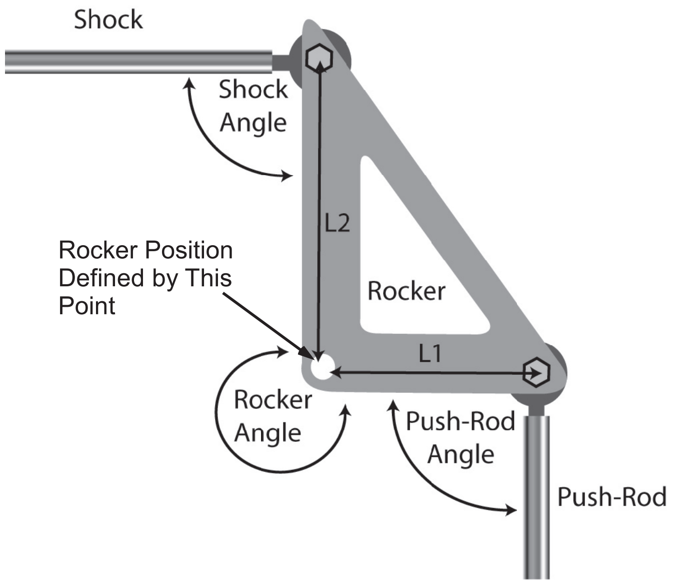


Figure 2.8: In this figure, the rocker, shock, and pushrod at ride height are illustrated. Important parameters that effect motion ratio are L1, L2, the shock angle, the push-rod angle, and the rocker angle

of the suspension when the car is sitting still with the driver seated. If these angles are 90 degrees at ride height, than it is reasonable to make the simplifying assumption that the motion of the pushrod and the shock are linear throughout the travel of the suspension. If the shock angle and the push-rod angle are not close to perpendicular at ride height, then this assumption is inaccurate. In this case the motion ratio is adversely effected. The way to control these two angles is to change the rocker angle.

Another important part of suspension design is that interference be avoided. Often, if components are moved or added to the car the suspension assembly must be changed. This causes the rocker position to be changed on a very frequent basis. Each time the rocker position is changed, it may result in a new rocker design.

The weight of any suspension component is particularly important because it contributes not only to the overall weight of the car but also to the unsprung weight of the suspension. As a result, it is desirable to use optimization on suspension components in order to make them as light as possible. At the same time, because the rocker is often moved, the optimal design of one day may not be optimal or even safe the next day. In order to use optimization on a rocker, the optimization must be able to withstand frequent changes to the objective functions because the loads to the rocker change when it is moved and the optimization must be able to withstand frequent changes to the constraints because each time it is moved the design domain changes.

2.5.2 Assembly

The first step towards an optimization problem that can survive changing objectives and constraints is to create a suspension subassembly that can provide all necessary inputs and receive any new outputs. In this case, it was necessary to create a suspension subassembly that could be easily adjusted to meet packaging requirements without compromising the motion ratio. The way this was accomplished was to allow the designer to select the position of the rocker as well as the lengths L1 and L2 on the rocker. Once these parameters were chosen, the rocker angle automatically adjusts to a value so that the shock and push-rod are perpendicular to the rocker at ride height. If a change in suspension parameters results in a change in the rocker than the next step is triggered.

2.5.3 Problem Statement

The optimization problem statement is slightly modified from the generic structural topology optimization statement.

$$\min_{x} J = \sum_{i=1}^{n} x_i v_i \tag{2.12}$$

subject to:

$$\delta \leq 0.01in \tag{2.13}$$

$$\sigma_i \leq 31183psi \quad i = 1,...,n_x \tag{2.14}$$

$$\Omega = \bigcup_{i=1}^{n} \Omega_i \quad i = 1,...,n_x \tag{2.15}$$

$$\Omega_i \cap \Omega_j = \emptyset, i \neq j \quad i,j = i,...,n_x \tag{2.16}$$

$$0 \leq x_i \leq 1 \quad i = 1,...,n_x \tag{2.17}$$

$$x_I = 1 \tag{2.18}$$

Each part of the optimization statement will be discussed as it is developed in this section.

2.5.4 Input Preparation

If a change in the rocker happens when the assembly is updated, than a macro calculates all input values needed to run topology optimization. The first step is to generate the new design domain Ω. The new L1, L2 and rocker angle are used along with established interfaces Ω_I between the rocker and the car to create a solid model that represents the new design domain. These interfaces are the pivot where it mounts to the frame, one bolt hole where it mounts to the shock, and one bolt hole where it mounts to the push-rod or pull-rod.

The second step is to calculate the load that is applied to the rocker. In order to accomplish this, a CATIA [14] macro inputs the new geometry into a spreadsheet which calculates the load case on the rocker when the suspension is fully compressed. Once these inputs are ready, they are exported to OptiStruct [15].

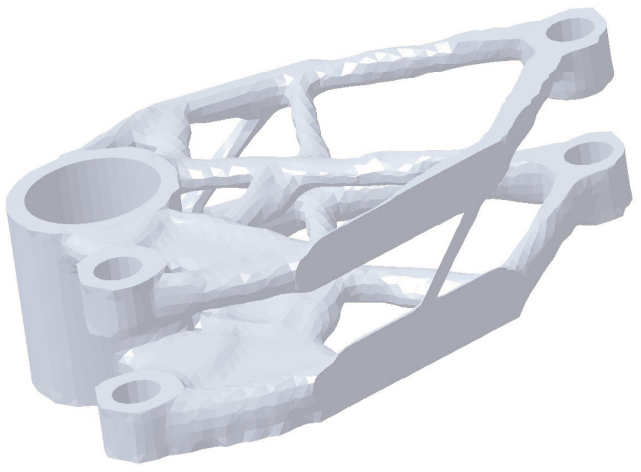


Figure 2.9: Optimized rocker arm for BYU Formula SAE car

2.5.5 Optimization Calculations

Once the design domain has been modeled and the load cases have been calculated the CATIA macro creates a text file with all of the information that OptiStruct needs to perform topology optimization on the model. An OptiStruct batch file meshes the design domain ,Ω, constrains the interface elements, Ω_I, applies the load cases, adds the stress constraint, σ, to each element and the deflection constraint, δ, to the point where the shock interfaces with the rocker, assigns the objective function, J, and runs the topology optimization. The results are output from OptiStruct to the assembly model where they can be verified and used if acceptable. An example of the topology optimization results can be seen in Fig. 2.9 compared to the original rocker design shown in Fig. 2.10. The optimization results were checked using Finite Element Analysis and both the stress and the deflection are within the constraints. The volume of the optimized rocker is 82% of the volume of the original design.

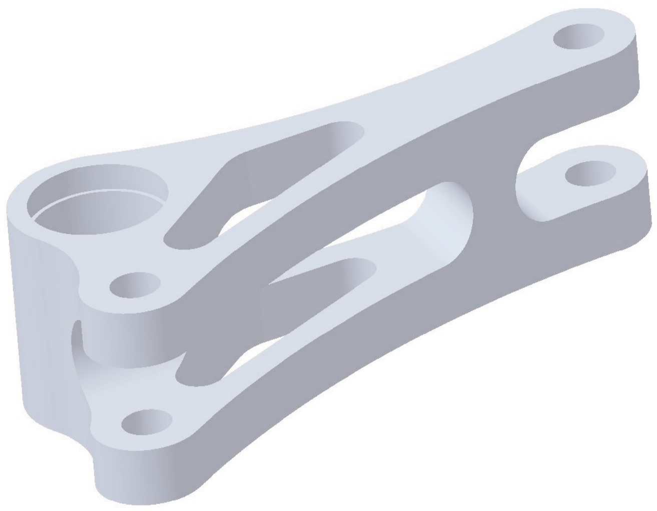


Figure 2.10: Original rocker arm for BYU Formula SAE car

2.6 Concluding Remarks

In this paper we have developed a design optimization framework that can be used when objectives, constraints, variables, and other conditions are expected to change as the design progresses and new information is gained. The design framework is parametric CAD and optimization based. By constraining the parametric updating of CAD models to meet optimization criteria, optimization methods are linked directly to changes that frequently occur during the development of a product. This link is fundamental to the developed framework because it facilitates the repeated optimizations of a part of the development of the system. In essence, if the CAD assembly is changed, the optimization is re-evaluated. One of the main benefits of the proposed method is that it reduces the designer's need to *react* to the design changes in one subsystem by manually correcting the affected design of another. The framework was demonstrated in the development of a suspension rocker for a BYU formula SAE car which was reduced in weight by 18%.

Chapter 3

Feasible-Configuration Generator for Multiple-Bend Springs

3.1 Description

This paper was published at the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference in 2006 by Christopher A. Mattson and K. Eric Bowman. The full reference for the paper is: Mattson, C. A., and Bowman, K. E., 2006. Feasible-Configuration Generator for Multiple-Bend Springs Tech. Rep. AIAA-2006-7094, 11th AIAA/ISSMO Multi-disciplinary Analysis and Optimization Conference, September.

3.2 Introduction and Literature Survey

Designers are continually adapting to the increasing market demand for smaller and smaller devices – particularly in the consumer electronics industry. One market requirement driving adaptation has a challenging effect on mechanical springs, such as those that comprise electronic connectors, locking latches, and switches. Specifically, the requirement is to miniaturize the spring size, given relatively unchanged requirements for spring deflections, normal forces, and maximum stresses [16]. To meet this requirement, some designers have identified feasible spring configurations that involve multiple bends in the spring geometry, such as the electronic contacts shown in Fig. 3.1.

The design of multiple-bend springs typically comprises four iterative parts: (i) a functional specification phase, (ii) a configuration generation phase, (iii) a configuration evaluation and selection phase, and (iv) a detailed design phase where shape and size optimization is performed for the selected configuration. The process of configuration generation fundamentally affects the success or failure of the design effort [17, 18, 19].

Specifically, by not generating potentially optimal configurations for further development in detailed design, the designer may unknowingly set the design on track for sub-optimal performance [20].

The developments presented in this paper focus on improving the important process of configuration generation. We define *feasible-configuration generation* in the context of progressive-die-formed springs as follows:

Feasible-Configuration Generation is the process of creating and/or identifying numerous spring outlines that can be used to feasibly satisfy designer-defined geometric and behaviorial requirements.

Often, during the conceptual design phase, the process of configuration generation and selection is executed on the sole basis of designer creativity, experience, and knowledge [20, 21]. As a result, the *feasibility* of a configuration is not typically evaluated in a rigorous way for numerous candidate configurations, rather it is often estimated based on conjecture [16].

In contrast, this paper presents an optimization-based spring configuration generator that rigorously checks the feasibility of the computationally generated configuration *before* presenting it to the designer as a candidate for evaluation. The proposed approach allows the designer to explore numerous optimization-generated spring configurations, which are generated specifically to feasibly satisfy his or her functional specifications. Importantly, the feasibility study is carried out *before* a final configuration is chosen for detailed development. Thus streamlining the designer's efforts to develop a design that has a high possibility of avoiding sub-optimality.

The notion of *feasible-configuration generation* is closely related to two notable research areas in the literature. They are; (i) concept generation [21, 22, 23], and (ii) topology or layout optimization [4, 24]. Similarities and differences between feasible-configuration generation and these two areas are briefly discussed in the remainder of this section.

Feasible-configuration generation and concept generation are similar in that (i) both processes are carried out separate from, and before, the important task of concept selection,

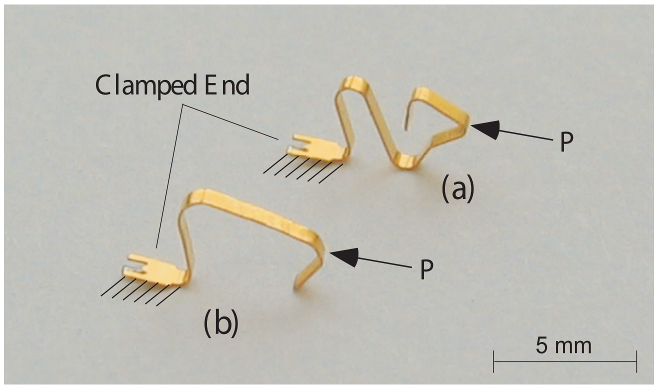


Figure 3.1: (a,b) Multiple-bend springs used as electronic contacts for hand-held computers.

and (ii) both processes seek to provide the designer with numerous designs possibilities – of which one (or a few) will be selected for further development in the detailed design phase.

One important distinction between these two methods is that feasible-configuration generation seeks to provide the designer with only feasible design possibilities. While traditional methods for concept generation are expected to yield roughly five percent working concepts; that is, concepts that are worth pursuing. These traditional methods include brainstorming [25], morphology [26], and synectics [27].

Importantly, we note that as the design space becomes more constrained, the traditional methods of concept generation would yield higher percentages of working concepts. The springs developed in this paper represent well-defined devices, and the concept generation thereof would result in much higher percentages of working concepts. It is this well-defined space that allows us to create a generic spring model that can be used together with optimization techniques to *ensure* that the generated configurations are feasible.

The feasible-configuration generator presented in this paper uses elements of both topological and shape optimization. It is similar to both optimization methods in that it uses numerical optimization to explore numerous shapes *and* topologies, respectively, where the geometry is the design variable. As shown in Section 3.4, the developments of this paper use a series of shape optimizations to explore the feasibility of disparate topologies.

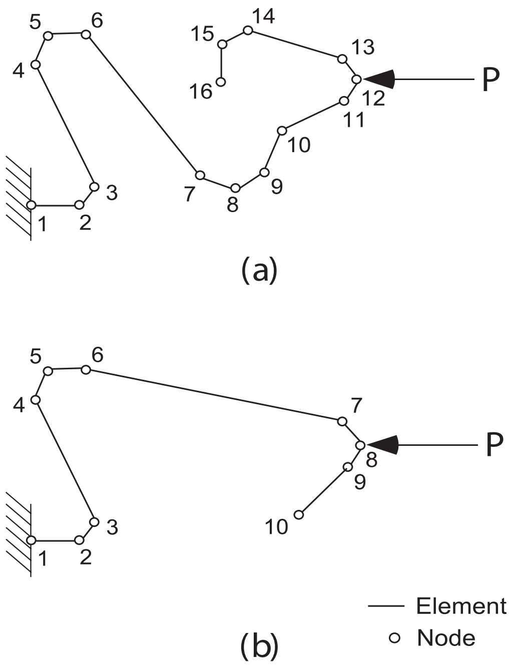


Figure 3.2: (a) Finite element model of spring shown in Fig. 3.1(a). (b) Finite element model of spring shown in Fig. 3.1(b)

One fundamental difference between feasible-configuration generation and topology optimization is that under the approach presented in this paper the final *physical implementations* of the generated spring configurations are meant to be and are topologically equivalent. We note, however, that before physical implementation – during the analysis phase – numerous disparate spring topologies are considered as part of the configuration generation process.

For example, consider the finite element model of the spring in Fig. 3.1(a) as it is shown in Fig. 3.2(a). This finite element model is used to evaluate the behavioral response of the spring when exposed to an applied load *P*. This finite element model is of a particular topology, while the spring model shown in Fig. 3.2(b) is of a different topology. That is, the finite element model in Fig. 3.2(b) cannot be transformed into that of Fig. 3.2(a) by pulling, stretching, twisting, bending, or squashing – without disassembling, adding elements, and/or reassembling elements [28].

Notice, however, that when we consider the physical implementation (see Fig. 3.1) of those two finite element models, we see that they *are* topologically equivalent. For the purpose of spring design, as its presented in this paper, the topological equivalence in the physical implementation is important because these springs will be formed in a progressive die; where raw material will be pulled, stretched, twisted, and bent to form the final shapes.

Another important difference is that under the approach presented in this paper, the topology is not actively modified during a single optimization routine, as is the case with topology optimization. Rather, the topology is changed in between runs in a series of optimizations that are performed as part of the feasible-configuration generation process.

While similar to these methods, the approach presented in this paper is developed specifically for progressive-die-formed springs and benefits from the unique aspects presented in the following sections.

The remainder of this paper is organized as follows. In Section 3.2, we present technical preliminaries and auxiliary developments. In Section 3.3 we develop the feasible configuration generator for progressive-die-formed springs. Following this section, in Section 3.4, we use the basic framework to generate numerous feasible spring configurations for an electronic contact. Finally, concluding remarks are provided in Section 3.5.

3.3 Technical Preliminaries and Auxiliary Developments

This section provides technical preliminaries and developments that facilitate the presentation of the feasible-configuration generator as introduced in Section 3.3. We first describe the modeling approach used to predict spring deflections and stresses. We then present the assumed design domain and the functional requirements typically associated with multiple-bend spring design. A general set of spring classes, which is developed to show diversity in the generated configurations, is then developed. Finally, we end this section by reviewing a generic multi-objective optimization problem statement as it will be used in the configuration generation algorithm.

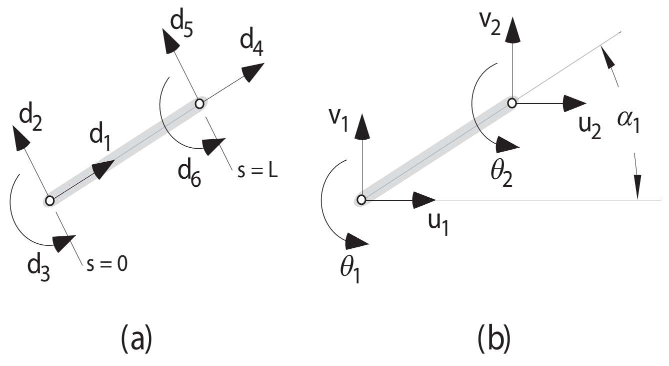


Figure 3.3: (a) Finite frame element in local coordinates. (b) Finite frame element in global coordinates.

3.3.1 Modeling Approach

The multiple-bend springs considered in this paper are modeled using linear elastic, isotropic, frame elements assembled together under a finite element methodology. The frame elements resist both bending and axial deformations [29], and are connected end-to-end starting at a fixed (clamped) end and continuing element-by-element to the free end of the spring. Each node in the finite element model is shared by a maximum of two elements.

Each element has six degrees of freedom ($d_1, d_2, ..., d_6$) as shown in Fig. 3.3(a). Figure 3.3(b) shows the six degrees of freedom as defined in the global coordinate system (u_1, v_1, θ_1, u_2, v_2, and θ_2). Global to local transformations are performed according to Eq. 3.1.

$$\left\{ \begin{array}{c} d_1 \\ d_2 \\ d_3 \end{array} \right\} = \left[\begin{array}{ccc} \cos\alpha_1 & \sin\alpha_1 & 0 \\ -\sin\alpha_1 & \cos\alpha_1 & 0 \\ 0 & 0 & 1 \end{array} \right] \left\{ \begin{array}{c} u_1 \\ v_1 \\ \theta_1 \end{array} \right\} \tag{3.1}$$

All elements in the structure have equal and constant rectangular cross-sections of width b, and material thickness h. The stiffness matrix for the generic frame element is

$$K = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{3.2}$$

where $I = bh^3/12$.

Static condensation is used to create the global stiffness matrix, $\mathbf{K}$. The nodal displacements are obtained by solving the system of linear equations given by

$$\mathbf{D} = \mathbf{K}^{-1}\mathbf{P} \tag{3.3}$$

where $\mathbf{D}$ is a column vector of global nodal displacements $(u_i, v_i, \theta_i, \forall i \in 1, 2, ..., n_n)$ where n_n is the number of nodes in the model, and $\mathbf{P}$ is a column vector of applied loads, expressed in global coordinates.

In this paper, we consider only the bending stress, σ, which is defined for the i-th element as

$$\sigma_i(s) = \frac{M_i(s)h}{2I} \qquad i \in 1, ..., n_e \tag{3.4}$$

where the bending moment is obtained by

$$M_i(s) = EI\frac{d^2 v_i(s)}{ds^2} \qquad 0 \le s \le L_i \tag{3.5}$$

and where the elemental lateral displacements (v_i) are found using Hermite interpolation functions [29] and are

$$v_i(s) = \begin{pmatrix} C_{1i} & C_{2i} & C_{3i} & C_{4i} \end{pmatrix} \begin{Bmatrix} d_{2i} \\ d_{3i} \\ d_{5i} \\ d_{6i} \end{Bmatrix} \tag{3.6}$$

where $C_{1i} = \frac{2s^3}{L_i^3} - \frac{3s^2}{L_i^2} + 1$, $C_{2i} = \frac{s^3}{L_i^2} - \frac{2s^2}{L_i} + s$, $C_{3i} = \frac{3s^2}{L_i^2} - \frac{2s^3}{L_i^3}$, and $C_{4i} = \frac{s^3}{L_i^2} - \frac{s^2}{L_i}$.

3.3.2 Design Domain, Key Parameters, and Variables

The design domain within which we will search for feasible spring configurations is illustrated in Fig. 3.4. The parameters depicted in the figure are defined in the following:

Design Domain ($\Gamma = [\Gamma_x \;\; \Gamma_y]^T$) is the two-dimensional space within which the search for feasible spring configurations will take place. Γ_x represents the length of the design domain in the cartesian-coordinate direction x. Likewise, Γ_y is the domain length in the y direction. We assume that the bottom left corner of the design domain is located at the origin (0, 0). Physically, this represents the window within which the spring design must fit.

Fixed End (x_1, y_1) is a designer-defined point in the design domain through which the fixed end of the spring is required to pass. All degrees of freedom at this point are equal to zero. Physically, this represents the point where the spring will be attached to a base structure, and is constrained by a clamped condition.

Free End (x_{n_n}, y_{n_n}) is a designer-defined point in the design domain through which the free end of the spring is required to pass when undeflected, where n_n is the number of nodes in the finite element model of the multiple-bend spring. All degrees of freedom at this point are free. Physically, this represents the point where the spring will make contact with another body, or the point of actuation.

Variable Point *i* ((x_i, y_i) where $n_n \geq 3$ and $i \in \{2 \ldots n_n - 1\}$) is a variable point in the design domain where a node in the finite element model of the spring will be located. The coordinates of this point (x_i, y_i) are design variables. Physically, this represents the point in the design domain through which the undeflected spring geometry will pass.

Desired Deflection ($\hat{\delta}_x$, $\hat{\delta}_y$) is the target deflected position of the free end. Note that the actual deflection is denoted by (u_{n_n}, v_{n_n}).

Load Magnitude and Direction (P, γ_P) is defined as applied to the free end. The load direction, γ_P, is always measured counter-clockwise from the horizontal, and P is

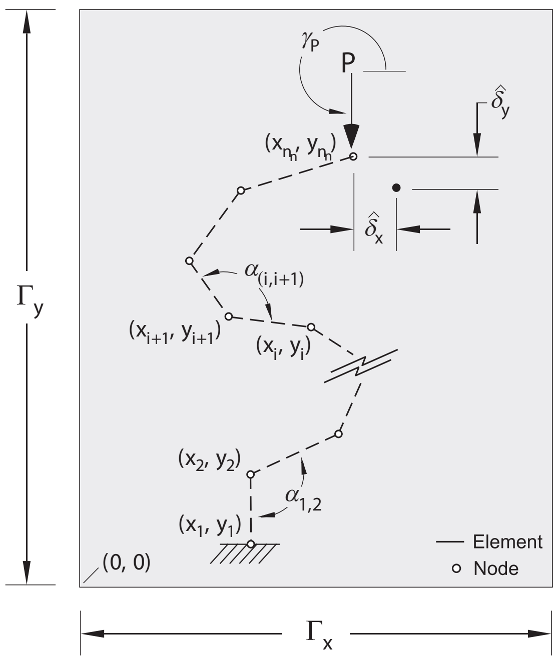


Figure 3.4: Assumed design domain Γ, key parameters, and variables.

always positive. Physically this can be thought of as the force and direction required to deflect the spring to the desired deflection.

Bend Angle ($\alpha_{i,i+1}$ where $i \in \{1...n_n - 1\}$) is the angle between adjacent elements and is always less than or equal to 180°. Physically this may become of particular interested when constructing manufacturing constraints.

3.3.3 Typical Functional Requirements for Springs

As discussed in the introduction, the design of multiple-bend springs typically comprises four steps. The first of which is the specification of functional requirements. For spring design, the designer would ideally be able to reach all of the design objectives listed below.

Ideally, the spring will;

1. Fit within a specified design domain.

2. Have a fixed end at a specified location.

3. Have a free end at a specified location.

4. Deflect to a desired position when *P* is applied.

5. Have a specified safety factor.

6. Be manufacturable.

In this paper we assume that the designer seeks to generate spring configurations that satisfy these requirements, and that any design that satisfies all of these requirements is of potential interest to the designer. As will be shown in the next section, multiobjective optimization can be used to identify numerous geometries that can feasibly meet the functional requirements listed above.

3.3.4 General Spring Classes

To demonstrate diversity in the generation of spring configurations, we must first develop a spring classification system, which we do here. All springs considered in this paper can be classified using the three digit classification number illustrated in Fig. 3.5.

The first digit in the classification number represents the direction that the first element leaves the node at the fixed end. The second digit represents the direction that the last element leaves the node at the free end. Eight directions are defined as shown in Fig. 3.6. The black solid circle in the center of the figure represents the node under consideration. The element leaving this node will leave in one of the eight directions shown. In Fig. 3.6, the element leaves the node in direction 8. Note that $67.5^\circ <$ Direction 1 $\leq 112.5^\circ$, $112.5^\circ <$ Direction 2 $\leq 157.5^\circ$, and so forth.

To be generally applicable, we use the direction of the applied load *P* as a reference direction. The configuration under consideration is rotated so that the applied load is 270 degrees counter clockwise from the horizontal.

The third digit in the classification number is the number of *curvature signs* the string of frame elements has from the fixed end to the free end. For example, consider the finite element model shown in Fig. 3.2(a). Nodes 1-4 represent the first curvature sign. We can observe an inflection point between nodes 2 and 5. Here, nodes 4 to 7 represent the

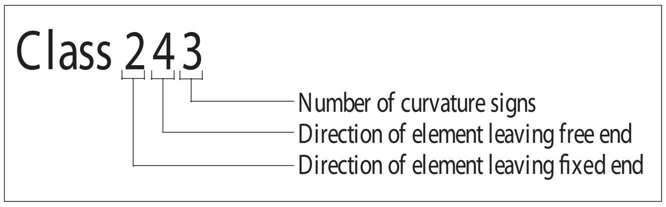

Class 2 4 3
Number of curvature signs
Direction of element leaving free end
Direction of element leaving fixed end

Figure 3.5: Configuration classification system.

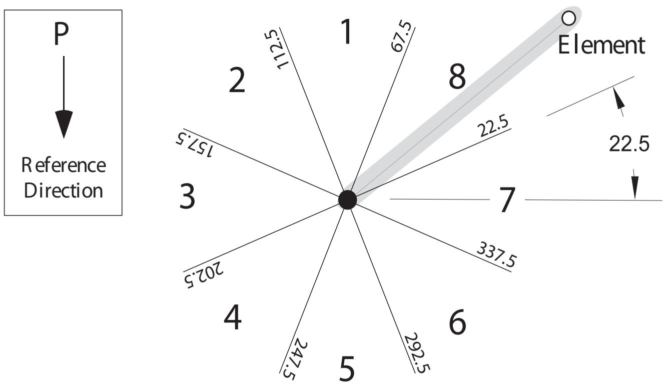


Figure 3.6: Eight basic directions from which an element can leave a node.

second curvature sign. Continuing to the end we observe three other inflection points. Thus there are 5 curvature signs in this configuration. As a note, a straight line between the fixed and free ends has no curvature; in this case the third digit is zero.

Continuing to examine Fig. 3.2(a), we Rotate the configuration so that the applied load is directly downward, and observe that the element leaving the fixed end leaves in direction 1. Also, the element leaving the free end (node 16) leaves in direction 3. Thus, this configuration belongs to Class 135. For comparison purposes, the configuration in Fig. 3.2(b) belongs to Class 122.

3.3.5 Generic Multi-Objective Optimization

The configuration generation algorithm presented in the next section uses multi-objective optimization to search for numerous spring configurations that satisfy all of the designer's functional requirements. Here, we provide a brief description of the generic multiobjective optimization problem statement as presented in Problem 1.

Problem 1: Generic Multi-objective Optimization

$$\min_{x} \; [\; \mu_1(x) \;\; \mu_2(x) \;\cdots\; \mu_{n_o}(x) \;]^T \tag{3.7}$$

subject to

$$g(x) \leq 0 \tag{3.8}$$

$$h(x) = 0 \tag{3.9}$$

$$x_{il} \leq x_i \leq x_{iu} \qquad i = 1,...,n_x \tag{3.10}$$

where μ_i represents the i-th design objective, n_o is the number of design objectives, g and h are inequality and equality constraint vectors, respectively, and x is a vector of design variables.

The process of aggregating multiple objectives into one function to be minimized is an important topic [30]. For the configuration generator presented in the next section, we use a compromise programming [31] approach to the aggregation. Specifically,

$$f = w_1\mu_1^m + w_2\mu_2^m + \cdots + w_{n_o}\mu_{n_o}^m \tag{3.11}$$

where we set w_1, w_2, $\cdots$, $w_{n_o} = 1$, and m equal to a positive even number.

3.4 Development of the Feasible Configuration Generator

In this section we present an iterative six-step process for generating feasible spring configurations using multiobjective optimization. In each iteration of the process, a shape optimization is performed wherein the shape of the spring is nearly completely free to move within the design domain. The topology of the finite element model used in the analysis and the objective function formulation may also change between iterations, thus expanding the search for feasible configurations.

Performed iteratively, the following six steps can result in the generation of numerous spring configurations that satisfy a designer-established functional specification as defined in Section 3.2. Each of the six steps in the process are discussed below. The iterative nature of the process is illustrated in Fig. 3.7.

STEP 1: SPECIFY REQUIREMENTS. Specify the fixed parameters: Γ_x, Γ_y, x_1, y_1, x_{n_n}, y_{n_n}, $\hat{\delta}_x$, $\hat{\delta}_y$, P, γ_P, $\alpha_{\min}$, $L_{\min}$, E, σ_y, F_s, b, and h; where $L_{\min}$ is the minimum element length, E is the modulus of elasticity, σ_y is the yield strength, and F_s is the stress safety factor. These parameters provide direction for the feasible-configuration generation process. When these requirements are satisfied, the resulting configuration is considered to be of potential interest to the designer.

STEP 2: SPECIFY GENERATOR PARAMETERS. The search for feasible spring configurations considers finite element models comprising two nodes to as many as N number of nodes. Therefore, the configuration generator parameter N must be specified. We will, in general, allow n_i to be defined as $n_i = 2, 3, ..., N$. Also, the search will consider one objective function formulation to as many as M formulations. Therefore, the generator parameter M must also be initialized. Specifically for the case where compromise programming is used, we allow m_k to be defined as $m_k = 2, 4, ..., M\text{x}2$.

ALGORITHM STEP: For each $n_i \in \{2, ..., N\}$ and for each $m_k \in \{2, 4, ..., (M\text{x}2)\}$ perform steps 3 through 6 as presented below. Each cycle of steps 3 through 6 results in either (i) computationally identifying a geometric configuration that satisfies the functional requirements established in step 1, or results in (ii) not finding a feasible solution.

STEP 3: INITIALIZE VARIABLES. Initialize the design variables x and y

$$x = \left[\begin{array}{cccc} x_2 & x_3 & \cdots & x_{(n_i-1)} \end{array} \right]^T \tag{3.12}$$

$$y = \left[\begin{array}{cccc} y_2 & y_3 & \cdots & y_{(n_i-1)} \end{array} \right]^T \tag{3.13}$$

where the initial values for these variables are determined as follows. Consider the vector, $\vec{w}$, whose tail is at the fixed-end node and head is at the free-end node. An evenly distributed set of n_i nodes are created along this vector. For the configuration geometry defined by

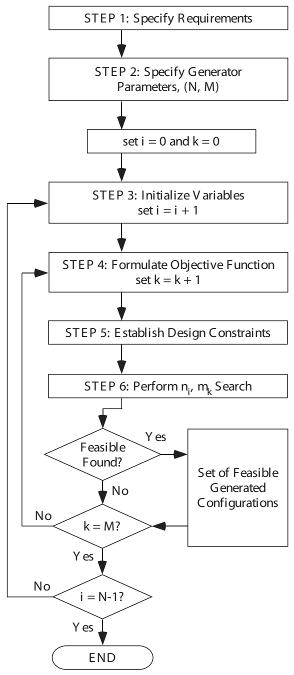

STEP 1: Specify Requirements
STEP 2: Specify Generator Parameters, (N, M)
set i = 0 and k = 0
STEP 3: Initialize Variables set i = i + 1
STEP 4: Formulate Objective Function set k = k + 1
STEP 5: Establish Design Constraints
STEP 6: Perform n_i, m_k Search
Feasible Found?
Yes
Set of Feasible Generated Configurations
No
No
k = M?
Yes
No
i = N-1?
Yes
END

Figure 3.7: Feasible configuration generator flow diagram.

nodes one through n_i, the design variables are the x and y nodal coordinates for nodes two through $n_i - 1$. Specifically,

$$\vec{w} = \begin{bmatrix} x_{n_i} & y_{n_i} \end{bmatrix} - \begin{bmatrix} x_1 & y_1 \end{bmatrix} \tag{3.14}$$

for $j = 2,3,...,n_i - 1$

$$\begin{bmatrix} x_j & y_j \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \end{bmatrix} + \frac{j-1}{n_i - 1}\vec{w} \tag{3.15}$$

Note that the initial values for the design variables define a straight line of elements that would be defined as belonging to Class AB0, where A and B would be based on the locations of the fixed and free-end nodes, and 0 classifies the straight line.

STEP 4: FORMULATE OBJECTIVE FUNCTION. As different objective functions will result in potentially different optima, we allow the objective function formulation to change during the feasible-configuration generation process. In this paper, we take a compromise programming approach and formulate the objective function as

$$f(x,y) = (\hat{\delta}_x - u_{n_i})^{m_k} + (\hat{\delta}_y - v_{n_i})^{m_k} \tag{3.16}$$

where the compromise programming power m_k is changed as steps 3 through 6 are cycled through (see Algorithm Step above).

STEP 5: ESTABLISH DESIGN CONSTRAINTS. For each of the finite element model topologies considered, we create optimization design constraints based on the functional requirements established in step 1. Specifically, we limit the bending stress in each element to

$$\sigma_j \leq \frac{\sigma_y}{F_s} \quad j = 1, ..., n_e \tag{3.17}$$

where n_e is the number of elements ($n_e = n_i - 1$).

Also, we prevent bends that are unduly difficult to manufacture by limiting element length and the angle between adjacent elements to

$$L_j \geq L_{\min} \quad j = 1, ..., n_e \tag{3.18}$$

$$\alpha_j \geq \alpha_{\min} \quad j = 1, ..., (n_e - 1) \tag{3.19}$$

Finally, the design variables, x and y, are permitted to move freely within the design domain by setting

$$0 \leq x_j \leq \Gamma_x \quad j = 2, ..., (n_i - 1) \tag{3.20}$$

$$0 \leq y_i \leq \Gamma_y \quad i = 2, ..., (n_i - 1) \tag{3.21}$$

Note that x_1, x_{n_i}, y_1, and y_{n_i}, are constrained by the functional requirements established in step 1.

STEP 6: PERFORM SEARCH. We now execute the shape optimization described in Problem 2_i^k for the topology defined by n_i and the objective function defined by m_k.

Problem 2_i^k: Shape Optimization for Topology Defined by n_i and Objective Function m_k

$$\min_{x,y} f(x,y) = (\hat{\delta}_x - u_{n_i})^{m_k} + (\hat{\delta}_y - v_{n_i})^{m_k} \tag{3.22}$$

subject to subject to Eqs. 3.17–3.21.

At the completion of the iterative execution of the six-step process developed here, the algorithm will have completed the search for feasible spring configurations by consid-

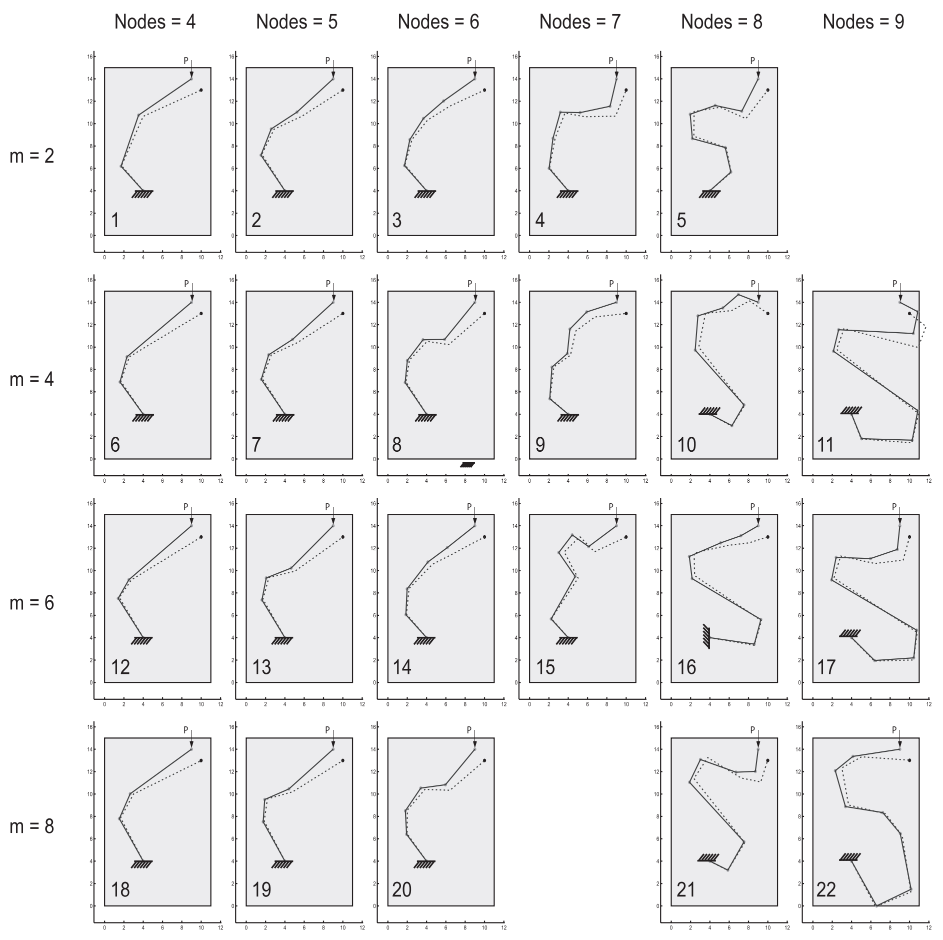


Figure 3.8: Twenty-two feasible contact configurations were edentified, for the electronic contact example, using the proposed feasible-configuration generator. These feasible configurations represent eleven different classes.

ering *N* different finite element model topologies, *M* different objective function formulations, and having performed $(N-1)\text{x}M$ shape optimizations.

3.5 Example: Configuration Generation of an Electronic Contact

In this section we apply the general approach developed in the previous section to the design of an electronic contact. The example is presented in three sub-sections: we first present the problem context and objective; we then present the execution of the steps presented in the previous section; finally, we present the results and a brief discussion.

3.5.1 Problem Context and Design Requirements

Consider the design of a spring that will act as a conductive body to carry electronic signals from one device to another. The spring must be fixed to one device both electrically and mechanically at one end of the spring, and make separable mechanical and electrical contact with the mating device at the free end. To ensure a reliable electrical connection, the mating bodies must be pressed together with sufficient force to result in minimal electrical contact resistance. If designed properly, the spring will provide the force required to ensure electrical conductivity, given a physical mating of the two devices. The objective of this example is to show that we can generate various disparate contact configurations that meet the functional requirements (specified in the following paragraphs).

3.5.2 Execution of the Feasible Configuration Generator

The six-step process presented in Section 3.3 is now carried out for the contact design problem described above.

Step 1: This step calls for the establishment of functional requirements. According to the generic functional requirements for spring design, as described in Section 3.2, we assume that the following parameters are known by the designer: The design domain, the fixed-end position, the free-end position, the desired design deflections of the free end, the load (or reaction force) magnitude and direction at the free end, the material and cross section properties, and any other manufacturing constraints. Table 3.1 lists the parameter values used for the design of the electronic contact.

Step 2: In this step we specify the generator parameters N and M. For this example we consider topologies with as little as two nodes to as many as nine nodes. Therefore, we set $N = 9$. For the objective function formulation, we use the compromise programming powers of 2, 4, 6, and 8. Therefore, we set $M = 4$.

Step 3: This step requires that we initialize the design variables x and y. We construct the vector $\vec{w}$ as defined by Eq. 3.14 and generate $n_i \in \{2, ..., N\}$ number of evenly distributed nodes along the vector $\vec{w}$. The coordinates for the nodes between the fixed and free ends are starting positions for the variables x and y.

Table 3.1: Design requirements for electronic contact configuration generation example

Requirement	Symbol	Value
Design Domain (x)	Γ_x	11.0 mm
Design Domain (y)	Γ_y	15.0 mm
Fixed End Position (x)	x_1	4.0 mm
Fixed End Position (y)	y_1	4.0 mm
Free End Position (x)	x_{n_n}	9.0 mm
Free End Position (y)	y_{n_n}	14.0 mm
Desired End Deflection (x)	$\hat{\delta}_x$	1.0 mm
Desired End Deflection (y)	$\hat{\delta}_y$	-1.0 mm
Force at Free End	P	0.73 N
Direction of Applied Force	γ_P	270°
Safety Factor on Stress	F_s	1.5 min
Modulus of Elasticity	E	120.7e9 Pa
Yield Strength	σ_y	1.172e9 Pa
Cross-sectional width	b	1.8 mm
Cross-sectional height	h	0.25 mm

Steps 4 and 5: These steps involve formulating the optimization problem statements that will be executed in Step 6. The resulting problem statement is described in Problem 3.

Problem 3: Generic Optimization Problem Statement for Contact Configuration Generation Example

$$\min_{x,y} f(x,y) = (1.0 - u_{n_i})^{m_k} + (-1.0 - v_{n_i})^{m_k} \tag{3.23}$$

subject to

$$\sigma_j \leq 781 \text{ Pa} \quad j = 1, ..., n_e \tag{3.24}$$

$$L_i \geq 2.0 \text{ mm} \quad i = 1, ..., n_e \tag{3.25}$$

$$\alpha_i \geq 100^\circ \quad i = 1, ..., (n_e - 1) \tag{3.26}$$

$$x_1 = 4.0 \text{ mm} \tag{3.27}$$

$$x_{n_n} = 9.0 \text{ mm} \tag{3.28}$$

$$y_1 = 4.0 \text{ mm} \tag{3.29}$$

$$y_{n_n} = 14.0 \text{ mm} \tag{3.30}$$

$$0.0 \text{ mm} \leq x_i \leq 11.0 \text{ mm} \quad i = 2, ..., (n_n - 1) \tag{3.31}$$

$$0.0 \text{ mm} \leq y_i \leq 15.0 \text{ mm} \quad i = 2, ..., (n_n - 1) \tag{3.32}$$

where the nodal deflections and elemental stresses are obtained according to the modeling approach described in Section 3.2.

Step 6: In this step, Problem 3 is carried out to search for a feasible spring configuration given the topology and objective function formulation defined by n_i and m_k. Note that steps 3 through 6 are executed iteratively as shown in Fig 3.7. For this example, Problem 3 is solved a total of 32 times; once for each combination of m_k and n_i. These optimizations resulted in 22 specific design configurations from 11 different classes.

3.5.3 Generation Results

Figure 3.8 shows the generated feasible configurations. The shaded area represents the design domain. The solid lines are the pre-deflected geometries and the dashed lines are the deflected geometries. The solid circle at the end of the deflected spring is a plot of the desired design deflection. For context, we have provided Fig. 3.9 showing a possible physical implementation of feasible configuration number 4.

Table 3.2 provides the data for the generated configurations. Column 1 of Table 3.2 corresponds to the configuration number as shown in the bottom left corner of the design domains for the configurations in Figure 3.8. Column 2 is the class into which the configuration fits. Columns 3 and 4 are the actual (analysis) deflections that the configuration will undergo, when exposed to P. Column 5 is the RMS error between the target and actual defections at the free end; note that the units are mm. Column 6 is the safety factor on bending stress.

At this point in the design process, the designer has 22 feasible spring configurations that will meet his or her requirements as specified in step 1 of the generator. The important process of configuration evaluation and selection would now be carried out. Importantly, because the main functional objectives have been satisfied by the optimization based search,

Table 3.2: Results for electronic contact design

Config.	Class	u_n	v_n	Error	F_S
1	241	0.9995	-0.9999	0.0005	1.8
2	242	0.9993	-0.9987	0.0015	1.7
3	241	1.0012	-1.0013	0.0018	1.8
4	252	1.0001	-1.0003	0.0003	1.8
5	843	1.0007	-1.0005	0.0008	1.8
6	241	0.9994	-0.9997	0.0006	1.7
7	242	1.0006	-1.0007	0.0009	1.7
8	242	0.9990	-0.9983	0.0019	1.8
9	233	0.9997	-0.9987	0.0014	1.9
10	634	0.9995	-1.0003	0.0006	2.0
11	663	0.9992	-1.0001	0.0008	1.9
12	241	0.9994	-1.0016	0.0017	1.7
13	242	1.0021	-1.0016	0.0026	1.7
14	241	1.0024	-0.9973	0.0036	1.8
15	244	0.9737	-1.0203	0.0332	1.9
16	743	0.9975	-1.0020	0.0033	1.8
17	653	1.0019	-1.0005	0.0020	1.8
18	241	0.9718	-1.0279	0.0397	1.7
19	242	0.9846	-1.0182	0.0238	1.8
20	242	1.0092	-1.0130	0.0159	1.8
21	653	0.9760	-1.0023	0.0241	1.8
22	632	1.0162	-0.9778	0.0275	1.9

the designer can focus on selection criteria that were not modeled analytically. For example, the designer may wish to select the configuration that is the *most simple*, or one that satisfies an unmodeled constraint such as *not deflecting out of the design domain* (see configuration 11 in Fig. 3.8).

3.6 Concluding Remarks

In this paper we developed a feasible-configuration generator for multiple-bend, progressive-die-formed springs. The process of configuration generation can fundamentally affect the success of the design effort, especially if potentially optimal configurations are not at all generated. The purpose of the feasible-configuration generator presented in this paper is to provide the designer with feasible spring configurations during the spring

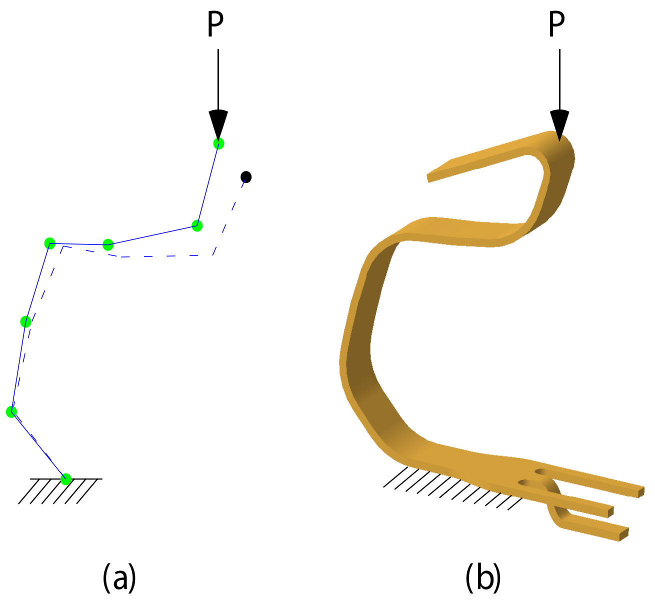


Figure 3.9: (a) Feasible configuration number 4 in isometric view. (b) Possible physical implementation of feasible configuration number 4.

conceptualization phase of the design process. In this way, the designer may consider a diverse set of feasible configurations before focusing on the detailed design of a select few. By combining principles from concept generation activities, and topology and shape optimization methods, we have developed a process that was able to generate 22 specific spring designs from 11 unique spring classes – all of which feasibly satisfy the functional requirements established for the design of an electronic contact.

Chapter 4

Practical Parametric CAD Based Optimization Framework with Iso-Performing Design Generation

4.1 Introduction

In Chapter 2, we explored a variety of concepts and methods that could help engineers use optimization to improve decision making during embodiment design. A framework that links optimization methods to parametric CAD was proposed in Section 2.4. The purpose of this framework is to allow engineers to invest time in an optimization model early in design with reduced the fear that later changes will invalidate the model. In Chapter 3, we explored optimization methods, classification schemes and viewing methods that could be used in embodiment design. The configuration generator created in Section 3.4 gives a designer feasible, iso-performing options (multiple designs with equal performance) to work with early in the design process. This helps the engineer explore the design space and gives insights into how to improve the design. In this chapter we describe a practical framework that further develops the theoretical framework with the idea of an *instantaneous optimum* and enhances the configuration generator with its concept of *iso-performance* by combining them in a CATIA V5 framework. We then demonstrate the utility of this framework by using it to design a Formula SAE frame.

The framework is embodied in a CATIA V5 toolbar that consists of seven commands. The commands listed in the order in which they should be used are: Create Node, Create Property, Create Element, Create Boundary Condition, Create Critical Displacement, GA Setup (with a the option to form either a pereto set or an iso-performance set), and Solve. These individual executibles combine into an application that fills the theoretical framework developed in Chapter 2. The theoretical framework is reproduced in Fig. 4.1.

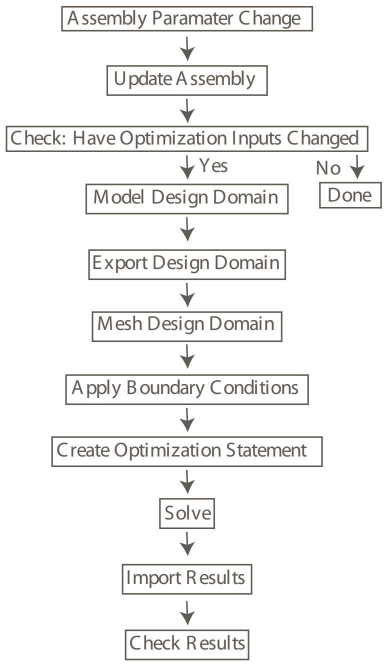

Assembly Paramater Change
Update Assembly
Check: Have Optimization Inputs Changed
Yes
No
Model Design Domain
Done
Export Design Domain
Mesh Design Domain
Apply Boundary Conditions
Create Optimization Statement
Solve
Import Results
Check Results

Figure 4.1: The parametric optimization process

The first steps of making an assembly parameter change and updating the assembly are common to parametric CAD, so in order accomplish them we based our application on the CATIA V5 API. Any command that creates CAD geometry does so through Visual Basic CATIA API calls that already exist. This allows the new Node and Element features to reside and be parameterized in the CATIA environment. The analysis and optimization portion are coded in object oriented Visual C++, and the two are linked through a combination of Microsoft Excel spreadsheets and input and output text files that emulate input and output decks from commercial FEA software.

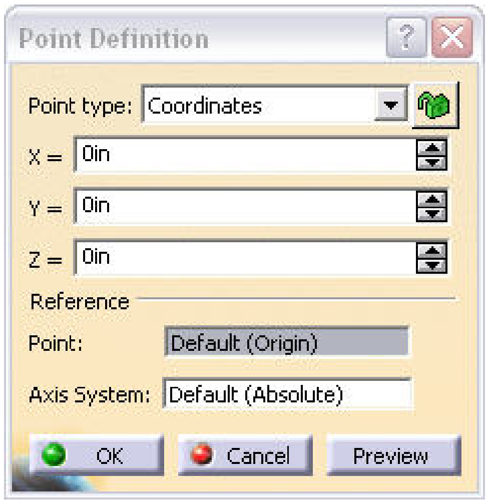
We decided to write our own analysis and optimization applications for a few reasons. The main reason that we chose to write our own algorithms is because we are adding functionality that is not available in commercial structural analysis and optimization softwares. The main feature that is not available in commercial software is the ability to create a single object that contains its geometric and structural definitions. An additional benefit from writing our own solver was increased speed. Because the kinds of problems our solver solves are limited, we can make a very specific solver that runs fast. The optimization algorithm is also be faster because it is specifically designed to do structural optimization by working directly with the analysis objects behind the scenes in C++.

The Create Node, Create Property, and Create Element commands fulfill the Model Design Domain and Mesh Design Domain steps. Because the geometry has the mesh definition included in it, the export design domain step is not necessary. Please note that these steps could be replaced with surfaces or solids associated with 2D or 3D meshes. The Create Boundary Condition command fulfills the apply boundary condition step. The Create Critical Displacement and GA Setup commands satisfy the create optimization statement step including the choice if the application should generate a pareto set or an iso-performance set, and the Solve command completes the solve step. The Solve command in this case uses finite element analysis and a genetic algorithm. These could be replaced with another kind of analysis and a different optimization algorithm. The import and check results steps are also based on the CATIA V5 API. The solve, import results, and check results commands in this application have the added ability to produce iso-performing designs similar to the feasible concept generator demonstrated in Chapter 3.

The remainder of the paper is organized as follows. In Section 4.2 we explore the fundamental components required to link CAD to topology optimization. Section 4.3 presents the model definition for automatic optimization-constrained updating of parametric CAD models. Finally the Chapter's concluding remarks are presented in Section 4.7.

4.2 Typical Optimization Process

The development of a new parametric modeling paradigm is complex, so it is helpful to break the project down into smaller tasks. In this section the different parts that link between parametric CAD and topology optimization are analyzed. Each task is outlined along with its important features. We begin by examining the basic traits of a modeling package (Sec. 4.2.1). We then consider meshing (Sec. 4.2.2), solving (Sec. 4.2.3), setting up the optimization problem (Sec. 4.2.4), the optimization algorithm (Sec. 4.2.5), and demonstrate an example of the application in use (Sec. 4.6) respectively.



Point type: Coordinates
X = 0in
Y = 0in
Z = 0in
Reference
Point: Default (Origin)
Axis System: Default (Absolute)

Figure 4.2: A standard CATIA point creation GUI.

4.2.1 Modeling

An image of the standard CATIA point creation graphical user interface (GUI) can be seen in fig. 4.2. A few important things to note about this point creation method are that it is intuitive, the user can change the origin from which the point is based, and the user can change the coordinate systems in which the point is made. Two additional and important features are the ability to associate any of the three point coordinates through parametric functions and a consistent dialog when creating, and later editing, the point. These features allow the user to create points in a variety of ways, edit them easily, and associate them directly with other geometry. Any new point creation commands should have a similar performance so that any link between design and analysis does not come at the expense of proper parametric design.

4.2.2 Meshing

Meshing 1-D elements is very straight forward from the user's end. They simply have to select a line or two points, and a 1-D element is formed. There is quite a bit, however, that goes on behind the scenes. Whenever a beam element is created, its stiffness matrix must also be created. The local member stiffness matrix ($\hat{k}$) of a beam element is formed as follows [32]:

$$\hat{k} = \begin{bmatrix} \hat{a} & \hat{b} & -\hat{a} & \hat{c} \\ \hat{b}^T & \hat{d} & -\hat{b}^T & \hat{f} \\ -\hat{a} & -\hat{b} & -\hat{a} & -\hat{c} \\ \hat{c}^T & \hat{f} & -\hat{c}^T & \hat{e} \end{bmatrix} \tag{4.1}$$

where

$$\hat{a} = \begin{bmatrix} \frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI_z}{L^3}(e_{z1}+e_{z2}+2e_{z1}e_{z2}) & 0 \\ 0 & 0 & \frac{3EI_y}{L^3}(e_{y1}+e_{y2}+2e_{y1}e_{y2}) \end{bmatrix} \tag{4.2}$$

$$\hat{b} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{3EI_z}{L^2}(e_{z1}+e_{z1}e_{z2}) \\ 0 & \frac{3EI_y}{L^2}(e_{y1}+e_{y1}e_{y2}) & 0 \end{bmatrix} \tag{4.3}$$

$$\hat{c} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{3EI_z}{L^2}(e_{z2}+e_{z1}e_{z2}) \\ 0 & \frac{3EI_y}{L^2}(e_{y2}+e_{y1}e_{y2}) & 0 \end{bmatrix} \tag{4.4}$$

$$\hat{d} = \begin{bmatrix} \frac{GJ}{L} & 0 & 0 \\ 0 & \frac{3EI_y}{L}(e_{y1}+\frac{e_{y1}e_{y2}}{3}) & 0 \\ 0 & 0 & \frac{3EI_z}{L}(e_{z1}+\frac{e_{z1}e_{z2}}{3}) \end{bmatrix} \tag{4.5}$$

$$\hat{e} = \begin{bmatrix} \frac{GJ}{L} & 0 & 0 \\ 0 & \frac{3EI_y}{L}(e_{y2}+\frac{e_{y1}e_{y2}}{3}) & 0 \\ 0 & 0 & \frac{3EI_z}{L}(e_{z2}+\frac{e_{z1}e_{z2}}{3}) \end{bmatrix} \tag{4.6}$$

$$\hat{f} = \begin{bmatrix} -\frac{GJ}{L} & 0 & 0 \\ 0 & \frac{2EI_y e_{y1} e_{y2}}{L} & 0 \\ 0 & 0 & \frac{2EI_z e_{z1} e_{z2}}{L} \end{bmatrix} \tag{4.7}$$

The member stiffness matrix in local coordinates is then multiplied by its respective transformation matrix to form the member stiffness matrix in global coordinates, or the members stiffness matrix in the global coordinate system:

$$R = \begin{bmatrix} X_x & X_y & X_z \\ Y_x & Y_y & Y_z \\ Z_x & Z_y & Z_z \end{bmatrix} \tag{4.8}$$

The member stiffness matrix in global coordinates for each member is assembled into a global stiffness matrix according to nodes. With a large finite element model, this matrix can become very large and time consuming to assemble. In a later section, one way to speed up the meshing process will be shown.

Forces at each node are assembled into a global force vector (F) and any applied or restrained displacements are assembled into a global displacement vector (U).

An undesirable aspect of most meshers is that first, line geometry must be created, after which it can be meshed. Once each line has been meshed, any points must be projected onto the mesh, and all mesh connections must be defined by hand. This is inefficient because a line has the same geometric definition as a beam element, and the end-points of the elements are coincident points. A unified modeling method that performs all these steps concurrently can save time.

4.2.3 Solving

Now that F, K, and U have been formed, the following equation can be made:

$$F = KU \tag{4.9}$$

Where F and K are known, and U is unknown.

One desired characteristic of a finite element solver is that it be fast. In some Finite Element Analysis textbooks [29], this equation is solved by simply inverting the stiffness matrix K and solving for U. This method can be very time consuming because matrix inversion is computationally very expensive at N_d^3 FLOPS where N_d is the number of degrees of freedom. In the optimization of large finite element models, solving the system of equations is the most time consuming part of the process. A faster method will be demonstrated later.

4.2.4 Optimization Problem Statement Definition

A general optimization problem statement is as follows:

$$\min_{x}\{\mu_1(x), \mu_2(x), ...\mu_n(x)\} \tag{4.10}$$

subject to:

$$g_q(x) \leq 0 \quad q = 1, ..., r \tag{4.11}$$

$$h_j(x) = 0 \quad j = 1, ..., v \tag{4.12}$$

$$x_{il} \leq x_i \leq x_{iu} \quad i = 1, ..., n_x \tag{4.13}$$

where each equation μ is a function of x, g is a set of r inequality constraints, h is a set of v equality constraints, and the design domain for each x is bounded by a lower bound, x_l, and an upper bound, x_u.

Each part of the optimization problem statement must be carefully formed. If this is not done properly, then the wrong equations will be solved, and the result is the wrong solution. This process is often prone to human errors. Streamlining the optimization problem statement formulation step would be very helpful.

4.2.5 Optimization Algorithm

There are two main criteria for an optimization algorithm, speed and flexibility. Usually, if speed is most important, gradient based algorithms may be used. Non-gradient based methods like simulated annealing and genetic algorithms do not converge as quickly as gradient based algorithms and have the added disadvantage of no optimality criteria.

Because non-gradient methods do not need gradient computations they can deal with discrete variable design spaces, noisy design spaces, and ill-posed design spaces better than gradient based methods. In these situations where flexibility is most important, non-gradient based algorithms should be used. In fact, in these situations gradient-free algorithms can converge to optimum solutions faster than newton or quasi-newton methods. Genetic algorithms have the added benefit of dealing very well with multi-objective design spaces because they can converge to a pareto frontier without carrying out a series of optimization runs.

4.3 Parametric Optimization Setup

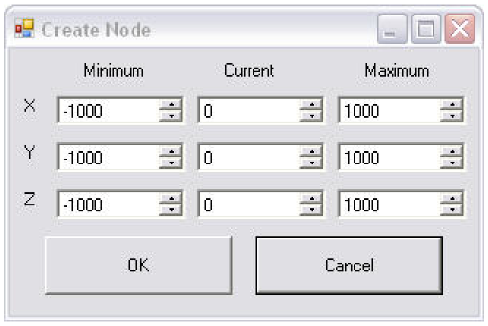
If an optimization problem is created in such a way that it can accept and adapt to change, then it could be used during conceptual design and be a valuable tool to help engineers make the best decision possible amongst uncertain or changing information. In this section we develop an optimization-constrained parametric CAD model definition that can help in this process.

4.3.1 Coupled Modeling, Meshing and Design Domain Definition

Meshing is often the most time intensive, most difficult step in the process of creating a finite element model from the user end. In order to keep this Thesis inside a reasonable scope, it does not propose being able to mesh any kind of geometry, which is reasonable since early designs are typically lower resolution than later ones. This Thesis develops methods for one kind of element, the beam element, that can later be applied to shell and solid elements. Since beam elements are very simple, it is reasonable to expect to be able to accomplish their geometric modeling, meshing, and design domain definition all at the same time.

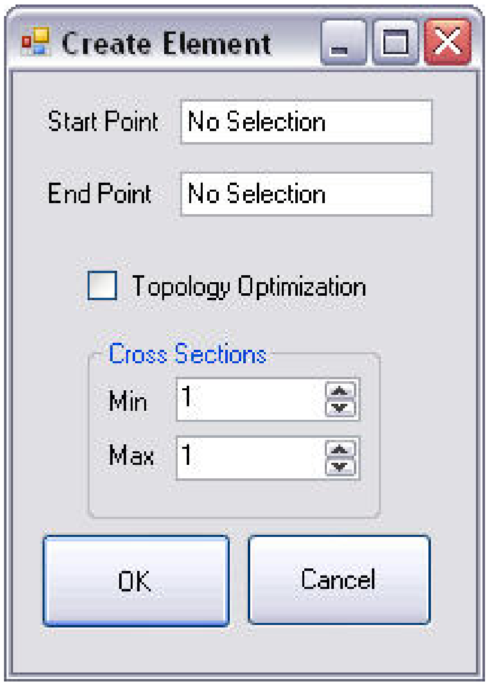
There are three main components that a beam topology optimization mesh requires. First, they require nodes. *Optimization-constrained nodes* can be created with the following GUI which is very similar to the point creation GUI in Fig. 4.3: With this GUI, when a point is created, it is automatically recognized as a node and its design domain for shape optimization is defined. This is the nature of an *optimization-constrained node*. Once nodes have been created, they can be linked with elements.

Second, elements must be defined between two nodes, just as a line is defined between two points. A beam element must have its cross sectional properties defined. In this case, cross sectional properties must be defined by the user with a GUI that will be shown shortly before an element can be created. The user can define which cross sections are



	Minimum	Current	Maximum
X	-1000	0	1000
Y	-1000	0	1000
Z	-1000	0	1000

Figure 4.3: Node creation GUI



Start Point: No Selection

End Point: No Selection

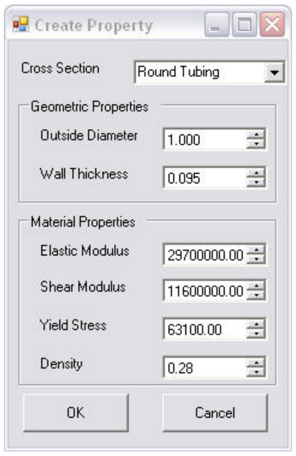
Min: 1

Max: 1

Figure 4.4: Element creation GUI

acceptable for use by changing the upper and lower cross section index. In this way, the line, the element, and the cross sections for size optimization are all defined concurrently as can be seen in Fig. 4.4 This GUI is very similar to the standard CATIA line creation GUI so CATIA users will already be familiar with how it works.

Finally, material properties must be defined. This can be done with the simple GUI in Fig. 4.5. It is patterned after the material properties command in CATIA Structural Analysis with a few enhancements. By using these three GUI's nodes, elements, and element


Create Property

Cross Section: Round Tubing

Geometric Properties

Outside Diameter: 1.000

Wall Thickness: 0.095

Material Properties

Elastic Modulus: 29700000.00

Shear Modulus: 11600000.00

Yield Stress: 63100.00

Density: 0.28

Figure 4.5: Property creation GUI

properties can be created with their respective geometries and optimization components in a direct, intuitive way.

4.3.2 Solving Efficiently in the Optimization Context

An important feature for the efficiency of this solver is that the solver does not use simple matrix inversion to solve the finite element system, instead it uses a modified Cholesky decomposition algorithm that requires only $\frac{N_d^3}{6}$ FLOPS [33]. The modified Cholesky algorithm is similar to LU decomposition in which any matrix A can be substituted with a lower and upper triangular matrix. LU decomposition will work on any square matrix. Cholesky's method, on the other hand, will only work for symmetric, positive definite matrices. Since all non-singular stiffness matrices are symmetric and positive definite, we can use Cholesky's method, which is faster. Modified Cholesky's method equates

$$A = LDL^T \tag{4.14}$$

where L is a lower triangular matrix with ones for each diagonal entry, and D is a diagonal matrix with zero's for all non-diagonal elements.

Once we have the lower and diagonal matrices, we can use back substitution to solve the system instead of matrix multiplication. This does not add or save any FLOPS beyond those saved by using modefied Cholesky decomposition; it is simply the way to solve a system that uses modified Cholesky decomposition.

$$KU = F \tag{4.15}$$

Now, set

$$K = LDL^T \tag{4.16}$$

So,

$$LDL^T U = F \tag{4.17}$$

We can substitute e for $DL^T U$ solve the following system with relative ease.

$$Le = F \tag{4.18}$$

following which we substitute d for $L^T U$ and solve the following set of equations.

$$Dd = e \tag{4.19}$$

finally, we solve the final system for the vector U.

$$L^T c = d \tag{4.20}$$

4.4 Genetic Algorithm

The genetic algorithm used in this Thesis is based on Richard Balling's Computer Structural Optimization [32]. The computer program proposed by Balling is presented in non-object oriented psuedo code based from C. Non-object oriented code of this kind does not integrate well with an object orietned API like CATIA V5's. So, a whole new object structure had to be created with C++ classes that was compatible with an object oriented API. In addition, care was taken to insure that the object structure would lend itself to organized user interaction.

4.4.1 Fitness

For each generation, each member of the generation is analyzed. The weight, deflection, and stress can be simultaneously considered. After each member has been analyzed, they are compared with a fitness function. This application uses the max-min fitness function [32]:

$$\max_{j \neq i} \left(\min_{k} \left(f_k^i - f_k^j \right) \right) \tag{4.21}$$

where f_k^i is the value of the kth objective of the ith design. This objective function gives good fitness to any pareto optimal design and gives better fitness values to those designs that are far separated from the rest of the group.

Inequality and equality constraints are handled in a similar manner. For an inequality constraint, the difference between the current value and the constraint limit is added to the fitness if the constraint is violated. For an equality constraint, the difference between the current value and the constraint value is always added to the fitness.

Diversity is also encouraged by penalizing similar designs. Each design is compared to the other designs before it in the current design array. The application counts how many of the elements between the two designs have matching cross sectional properties. The number of common cross sections with the most similar design is added to the fitness value, penalizing a design if it is very similar to another design with respect to cross sectional properties. Designs are not penalized for having similar nodal positions.

4.4.2 Tournament Selection with Elitism

After the fitness has been evaluated, the parents are selected for the next generation using tournament selection. First, the latest set of designs is combined with the best designs from all previous generations. Then, a user defined number of members are selected at random. The member selected with the best fitness is then chosen as a parent for the next generation. This is repeated until all parents are chosen.

4.4.3 New Generations

Once all parents have been chosen, they are combined to create the children for the next generation. First, crossover is performed, then mutation is performed on each variable of each design.

For crossover, a random number is generated and compared with a user defined crossover probability for each design variable. If the random number is larger than the crossover probability, then a random number r is generated. The variable r is used in the following blend crossover equations [32]:

$$y_1 = (r)x_1 + (1-r)x_2 \tag{4.22}$$

$$y_2 = (1-r)x_1 + (r)x_2 \tag{4.23}$$

where y_1 and y_2 are the new design variables from design one and design two respectively, and x_1 and x_2 are the old design variables from design one and two respectively. The advantage of this form of crossover over simply swapping design variables is that this method brings about smooth design changes rather than large design changes, which usually leads to faster convergence.

For mutation, a random number is generated and compared with a user defined mutation probability for each design variable. If the random number is larger than the crossover probability, then mutation is performed. Mutation is performed by replacing the design variable at hand with a random number within that variable's design domain.

4.4.4 Topology Optimization

Topology optimization is accomplished by allowing the cross sectional area of those members eligible for topology optimization to approach zero. Any element with near-zero cross sectional area can be ignored. Because the topology optimization is done by manipulating cross-sectional properties. Equal topologies are penalized by the fitness function described in Section 4.4.1

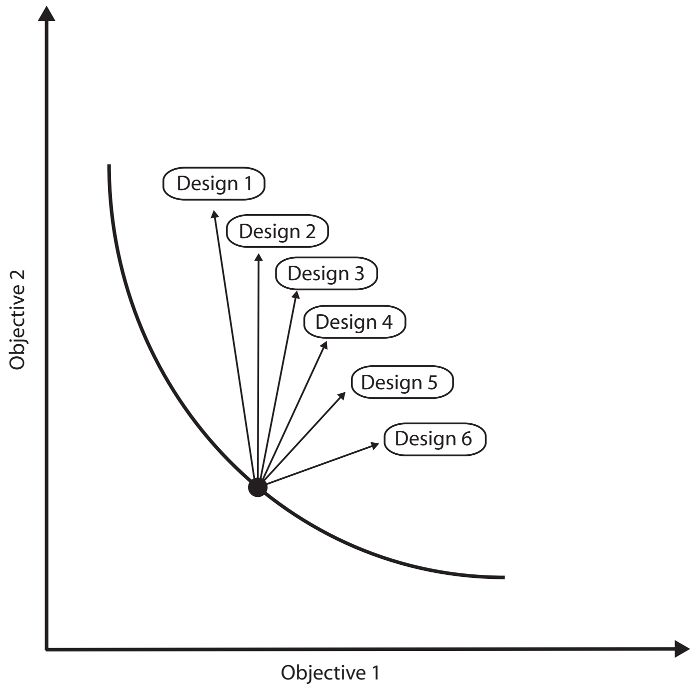


Figure 4.6: First find a pareto set, then find iso-performing designs around one design.

4.5 Iso-Performing Design Interpretation

This framework produces iso-performing designs by using equality constraints in the genetic algorithm while penalizing for duplicate topologies. With multi-objective optimization, instead of having only one optimal design, the engineer can be presented with a pareto set of optimal solutions. Selecting the appropriate configuration is often a difficult decision with little criteria to choose from. In the application created here, instead of forcing the designer to only keep one pareto optimal design, the application will keep the pareto frontier and let the designer change between different pareto optimal designs. Once a design has been chosen, the solver can produce a set of designs with similar performance, giving the engineer even more options with the desired characteristics. This concept is visually demonstrated in Fig. 4.6 where a pareto frontier is shown, along with an expansion on one point of that pareto frontier.

4.6 Formula SAE Frame

In this section, the optimization framework is used to design a Formula SAE chassis. It is demonstrated that the tools created speed up finite element model creation and optimization problem formulation. It is also demonstrated that this framework can deal with parametric changes driven by an outside source, and that it can produce iso-performing designs with a weight and stiffness of choice.

Using the programmed tools, it took 3 hours to create (1) a CAD model, (2) a finite element model, and (3) an optimization problem for the FSAE frame. In a similar situation it would take about 4 hours to create a CAD model and finite element model and another hour to set it up for optimization. However, discrete cross sectional properties could not be optimized, and topology optimization would not be possible. Both of which can be done with this framework.

The CAD mode mentioned above comprises points, lines, sweeps, and thickened surfaces that represent all major pieces of tubing in front of the engine. each of these lines is meshed with tubular steel cross sectional properties. Each is allowed to change to a variety of standard tubing sizes. Each member is only allowed to be as small as the minimum tubing required by FSAE rules. The engine mounts were then fixed in all translational directions and a mixed cornering, braking load case was applied to the suspension points for both a right and a left hand turn. The deflections minimized are those of the front upper right and left suspension pickup points. The value listed is the sum of the X, Y, and Z deflections of both the right and the left suspension point.

The genetic algorithm takes about 45 seconds per generation. It forms a pareto frontier in about 100 generations, or in about 90 minutes run time. 1000 generations produce a number of solutions that are pareto optimal along with the current design. If allowed to continue, the algorithm would eventually converge to the current design, but it cannot find a lighter solution than the current design because the current design is made to meet the bare minimum material requirements governed by race rules. It can, however, find designs that are much stiffer but are not much heavier. The initial population can be seen in Fig. 4.7. The pareto frontier after 250 and 750 generations can be seen in Fig. 4.8 and

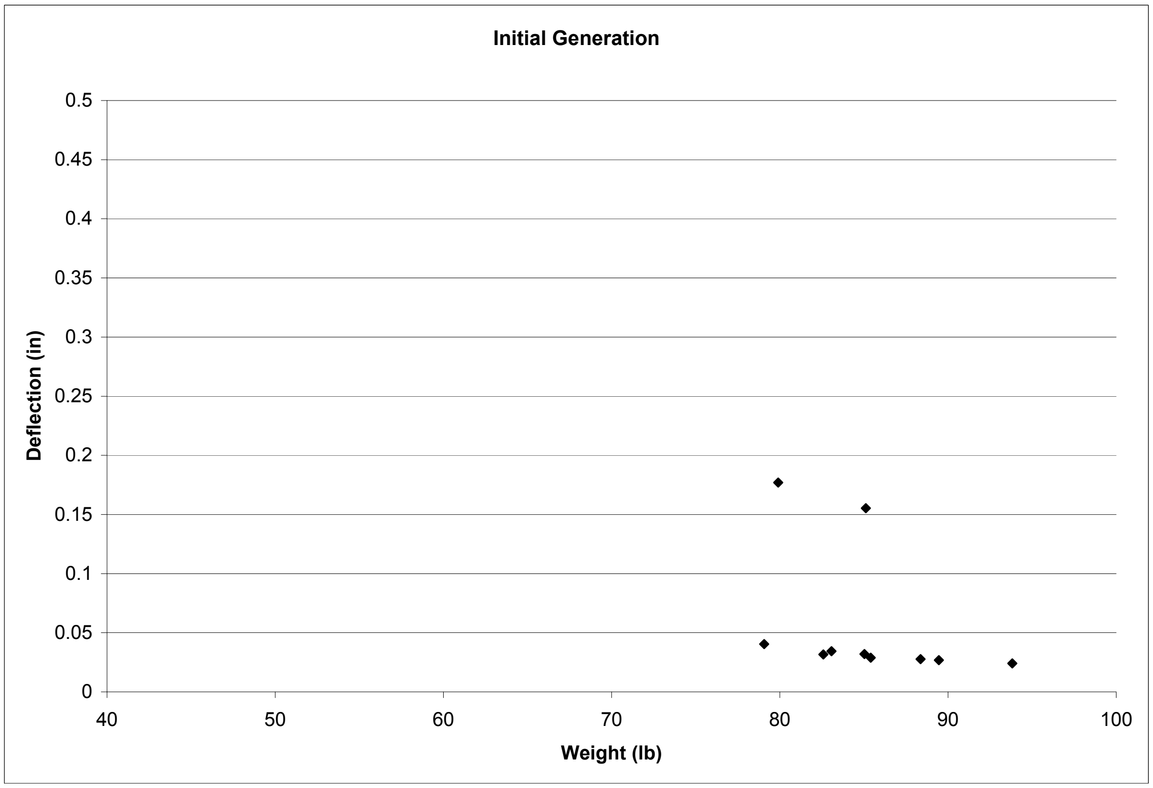


Figure 4.7: Initial population.

Fig. 4.9 respectively. The final generation after 1000 generations can be seen in Fig. 4.10. The Current design's performance is plotted as an open circle in Fig. 4.10.

The lightest design from the optimization run can be seen in Fig. 4.11 with its performance circled. The stiffest design can be seen in Fig. 4.12 with its performance highlighted in the same way. Fig. 4.13 and Fig. 4.14 show examples of compromise solutions that favor weight and stiffness respectively. Their performance is also circled.

In less time than it takes to make a geometric model, mesh it, and apply boundary conditions, an engineer can have a meshed model that has been optimized. The optimization algorithm returns not just one design, but a pareto set of designs that the engineer can use to become more familiar with the design space. The frames produced with this framework have counterintuitive design features that improve all of the designs from the lightest to the stiffest. In addition there are differences between the lightest and stiffest designs the engineer can learn from. Since the framework is integrated into a commercial CAD system,

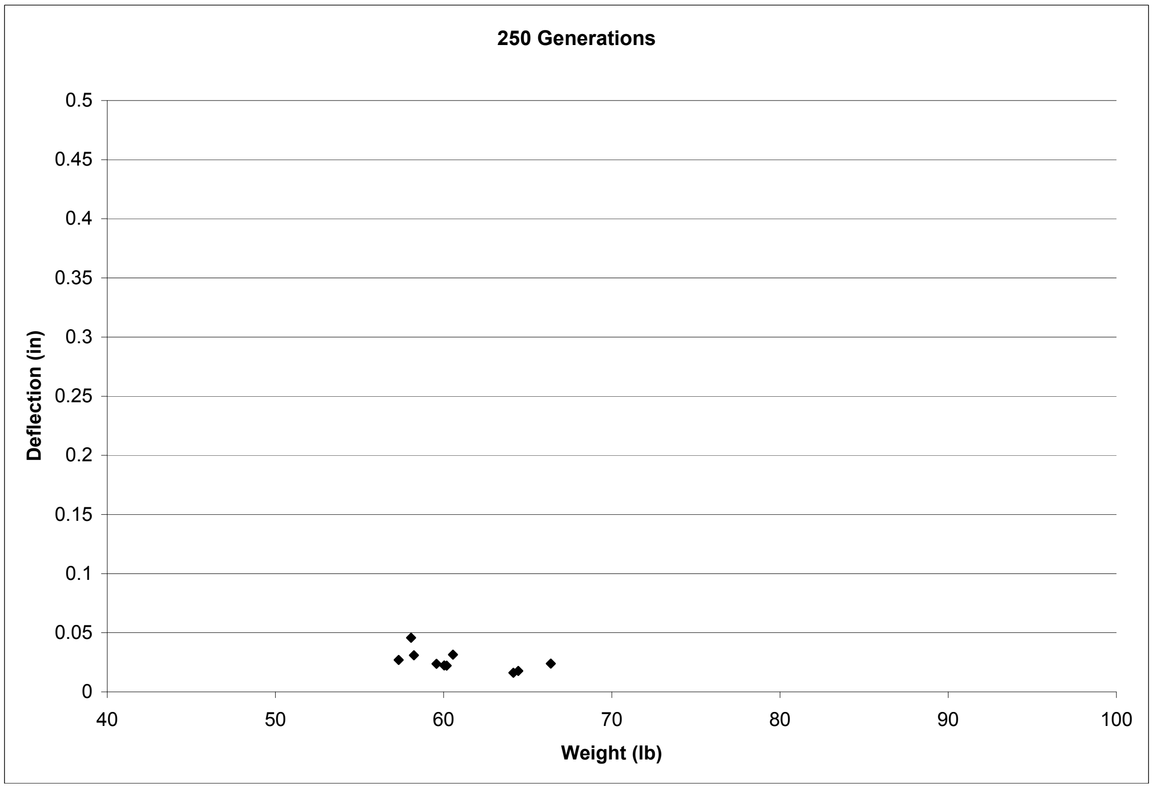


Figure 4.8: Design set after 250 generations

the designer can keep working with the results through the end of the design process. Next, we will demonstrate the utility of the iso-performance aspect of the framework.

In order to demonstrate the iso-performing design generation ability of the application, we chose to expand on a compromise design that favors weight. This design weighs 52.89 lbs. and has a deflection of 0.0669 in. Images of the final designs can be seen in Fig. 4.15. In addition, the performance of each design can be seen in Table 4.1.

An engineer can use the iso-performing configuration generator to further refine his or her knowledge of the design space at hand. Once the desired performance of a structure has been identified, the designer can produce multiple designs with the same performance. This can lead to the discovery of objectives and constraints that should have been included, or it could lead to a design that satisfies not only written constraints, but also unwritten constraints like aesthetics. The end result is not just one optimum that may or may not be satisfactory, but a set of optima that are much more informative. Next, we will demonstrate

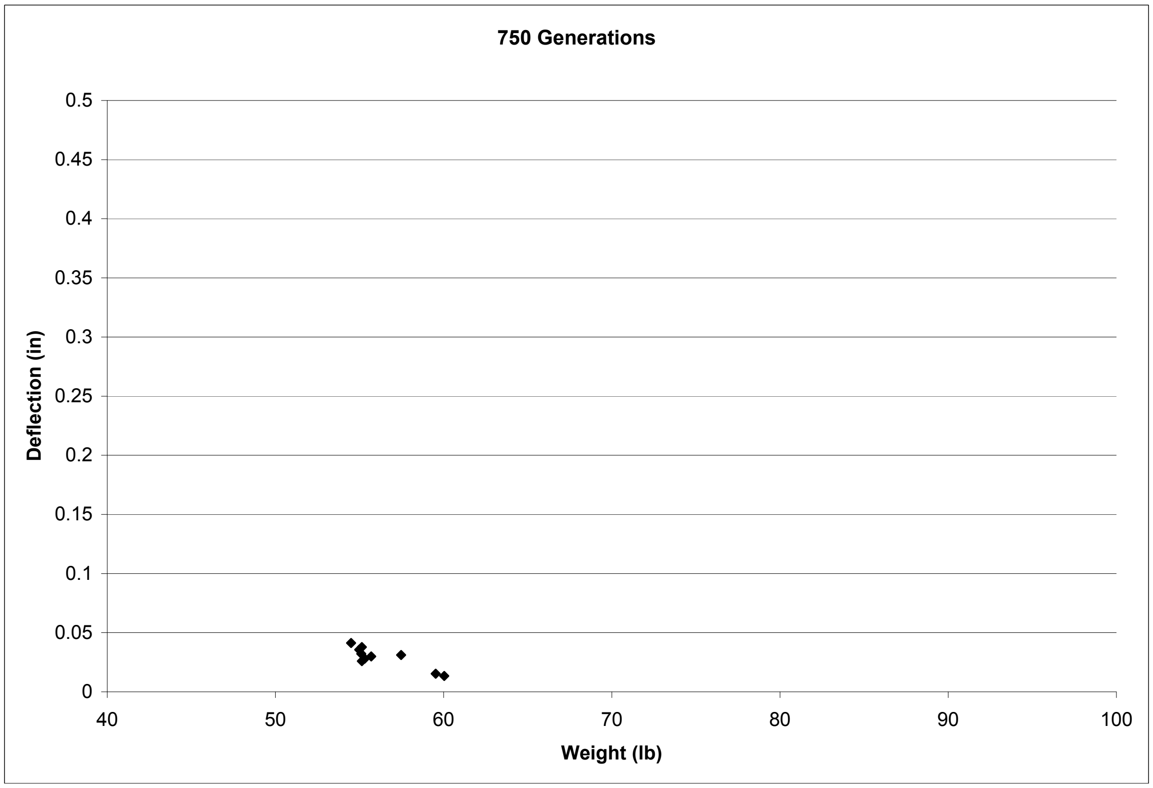


Figure 4.9: Design set after 750 generations

Table 4.1: Results for 1000 generation iso-performance optimization.

Config.	Weight	Deflection
1	52.82	0.0775
2	53.33	0.0897
3	53.74	0.0880
4	53.40	0.0665
5	54.11	0.0862
6	53.67	0.0907
7	53.42	0.0686
8	54.35	0.0875
9	52.97	0.1019
10	54.67	0.0856

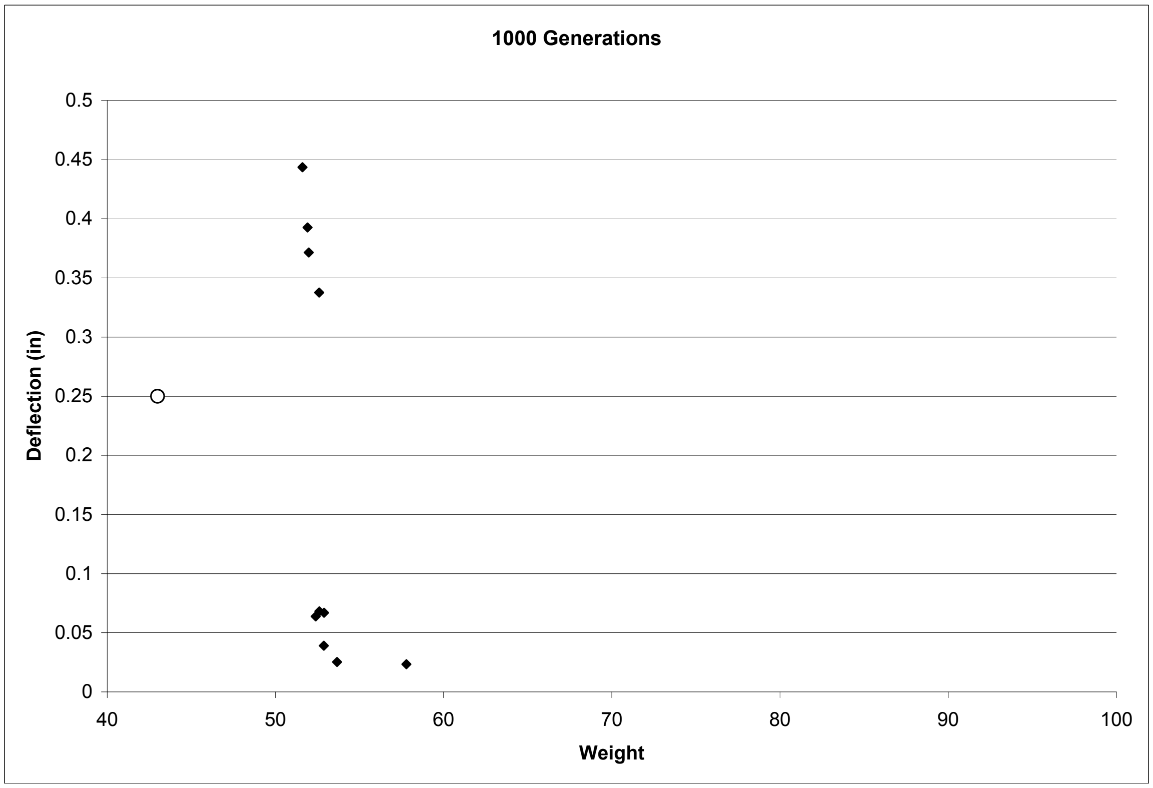


Figure 4.10: Design set after 1000 generations

how the optimization framework can be used to handle changes brought on by outside influences.

In order to demonstrate the ability of the framework to deal with parametric change, the suspension geometry was changed to a single-keel design. This means that the lower control arms nearly come to the centerline of the car as can be seen in Fig.4.16. Ten new iso-performing designs were found around the same design that weights 52.89 lbs. and has a deflection of 0.0669 in. This run consisted of 500 generations, showing that the algorithm at first converges very quickly, followed by slower progress. The design performances can be seen in Table 4.2. Note that in half the generations, the solutions are nearly as good as they were after 1000 generations.

This final feature of the optimization framework can allow designers to invest more effort into their geometric and analysis models early in the design process. Since the framework can react to standard parametric updates, the engineers can use standard parametric

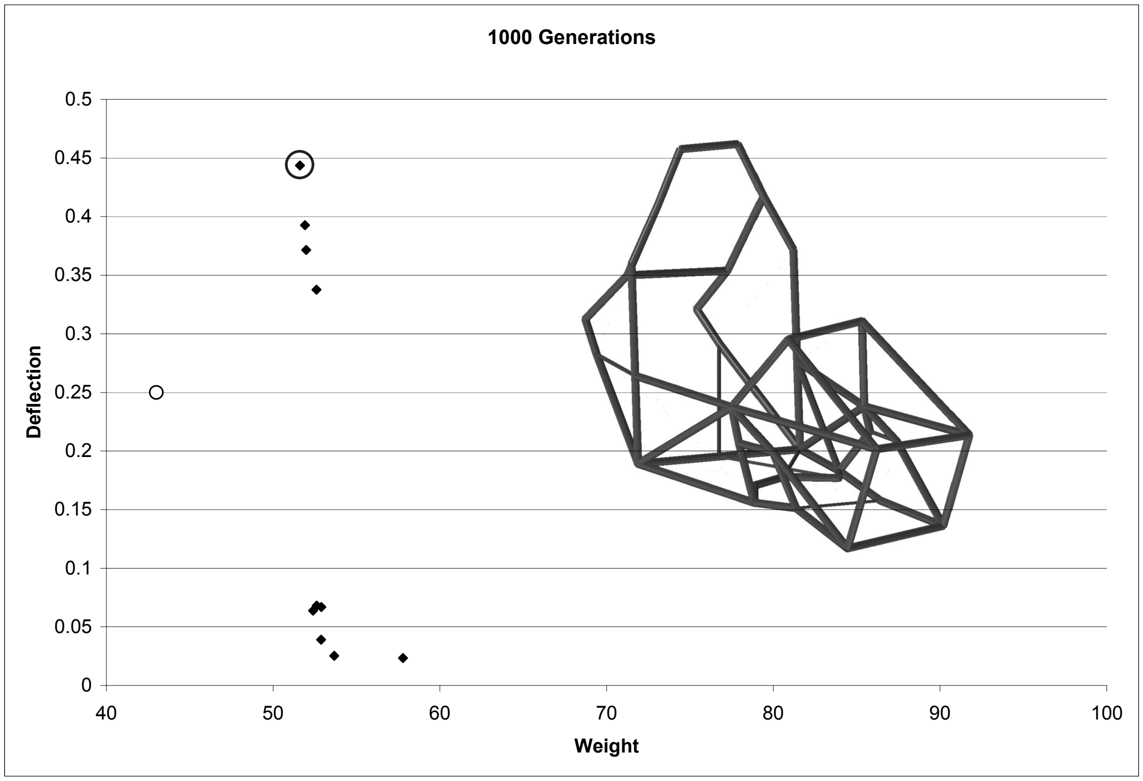


Figure 4.11: Lightest design after 1000 generations

Table 4.2: Results for 500 generation iso-performance optimization after a parametric change.

Config.	Weight	Deflection
1	54.69	0.0296
2	55.67	0.0233
3	55.91	0.0239
4	55.56	0.0279
5	55.68	0.0274
6	55.90	0.0282
7	56.41	0.0240
8	54.82	0.0407
9	55.32	0.0375
10	56.42	0.0279

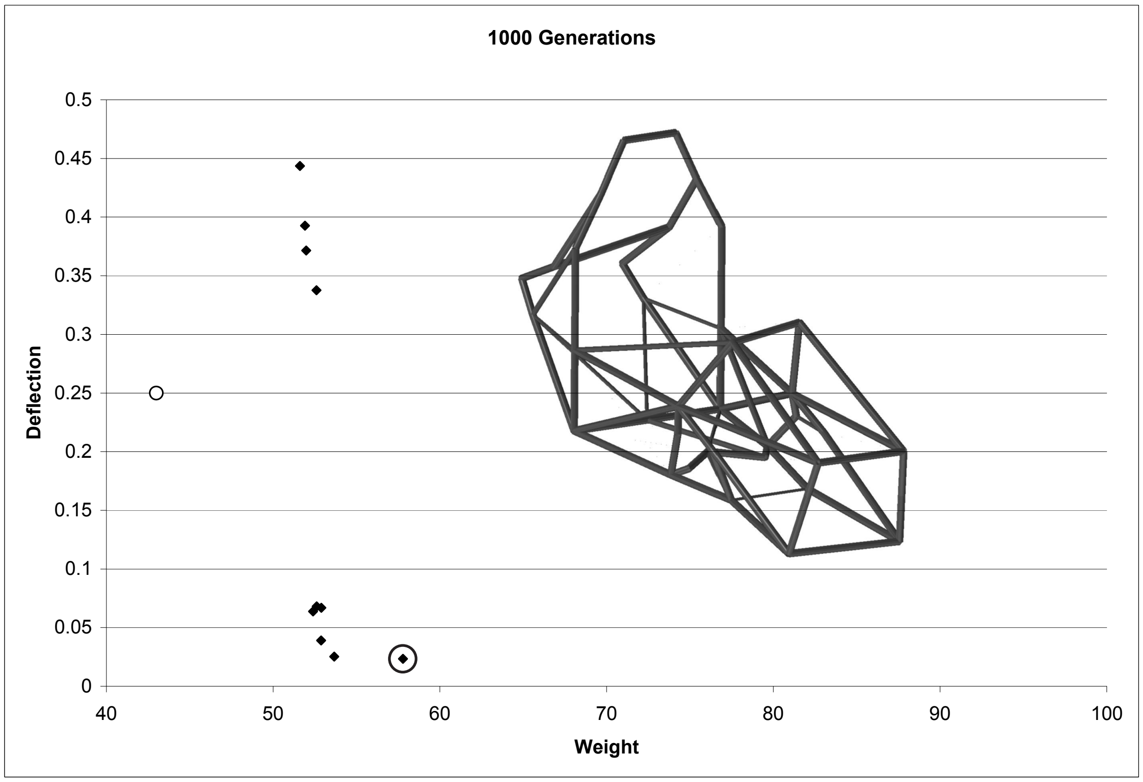


Figure 4.12: Stiffest design after 1000 generations

methods and take advantage of not just a geometric model update, but also of an optimization based update. Instead of being forced to react to changes from other groups, designers can focus on improving their model to include as much data as possible so that their final instantaneous optimum is as accurate as possible.

4.7 Chapter Summary

In this chapter we have developed an application that integrates the framework developed in Chapter 2 with the iso-performance concept presented in Chapter 3. This application has been integrated into CATIA V5 in such a way that a CATIA user experienced with line and point creation would be familiar with the geometry creation process. Additional commands necessary to complete a finite element model and optimization setup have also been created. This finite element solver takes advantage of Modified Cholesky's

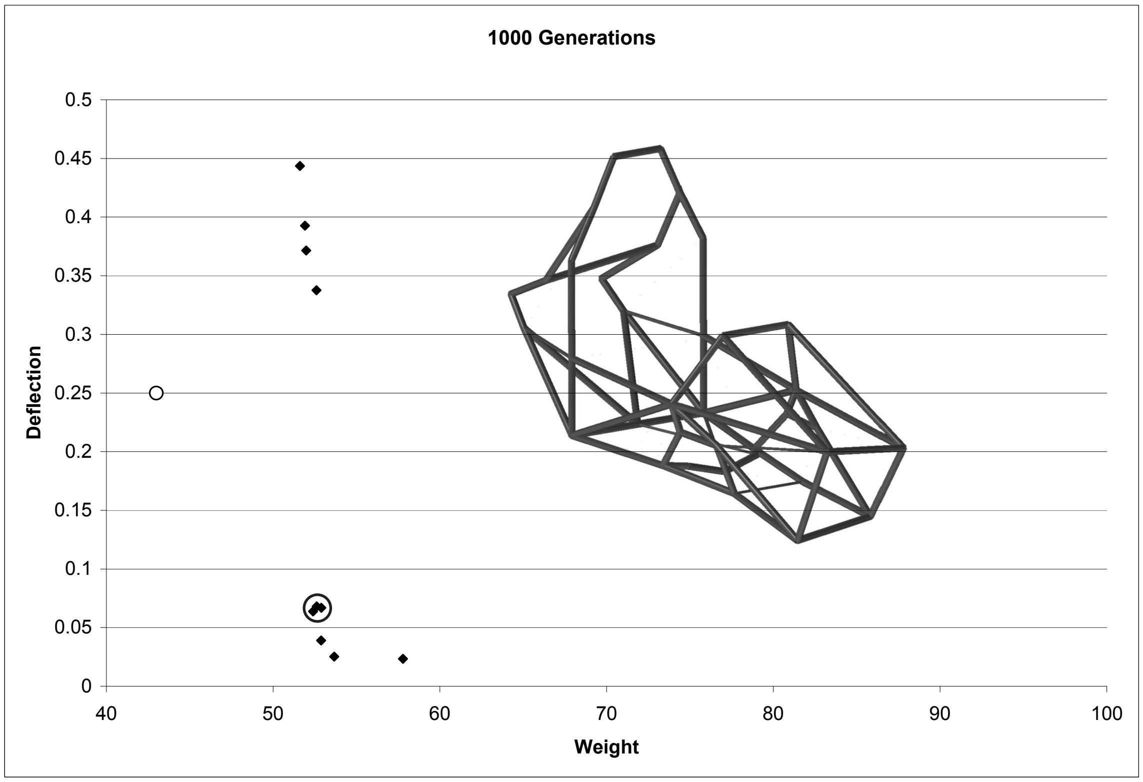


Figure 4.13: Compromise design favoring weight after 1000 generations

decomposition and removes constrained degrees of freedom in order to reduce runtime. A Genetic Algorithm that uses max-min fitness is used to develop pareto optimal designs and equality constraints are used to develop iso-performing designs. This application was then used to develop a FSAE frame. First, a pareto set of frames was found. Then, one design was selected and iso-peforming designs were found. Finally, a parametric change was made to the suspension geometry and a new set of frames with equal peformance to that found previously was found.

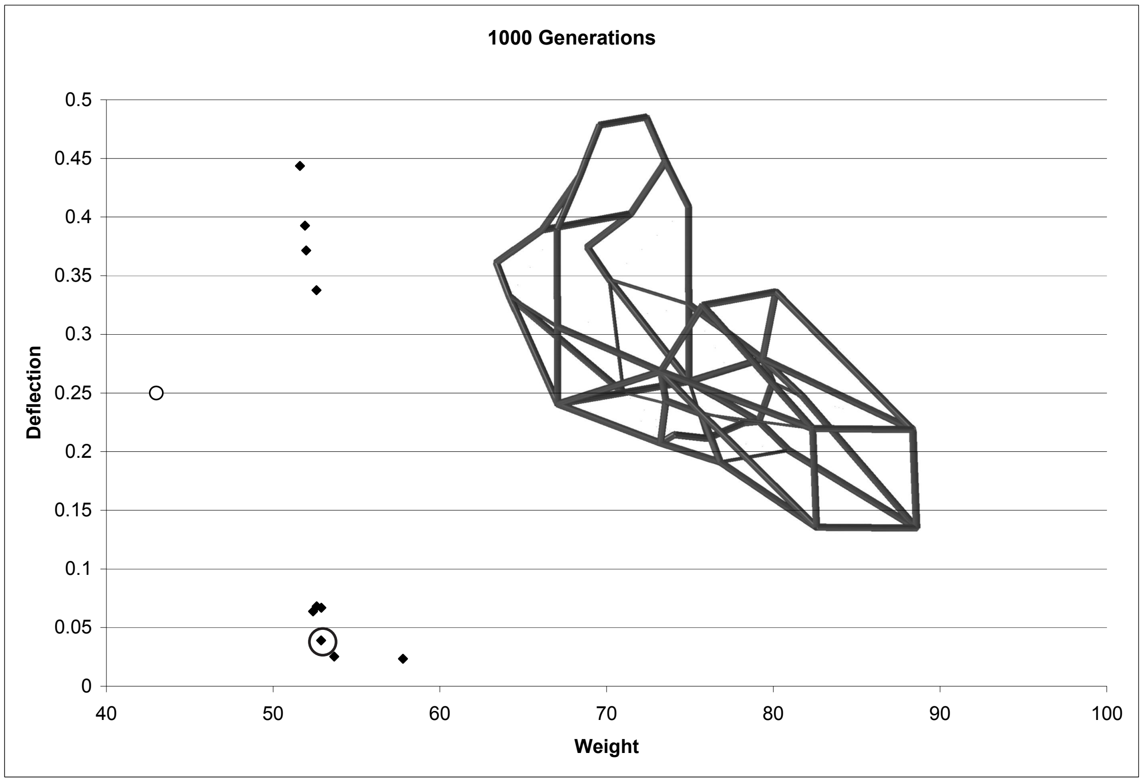


Figure 4.14: Compromise design favoring stiffness after 1000 generations

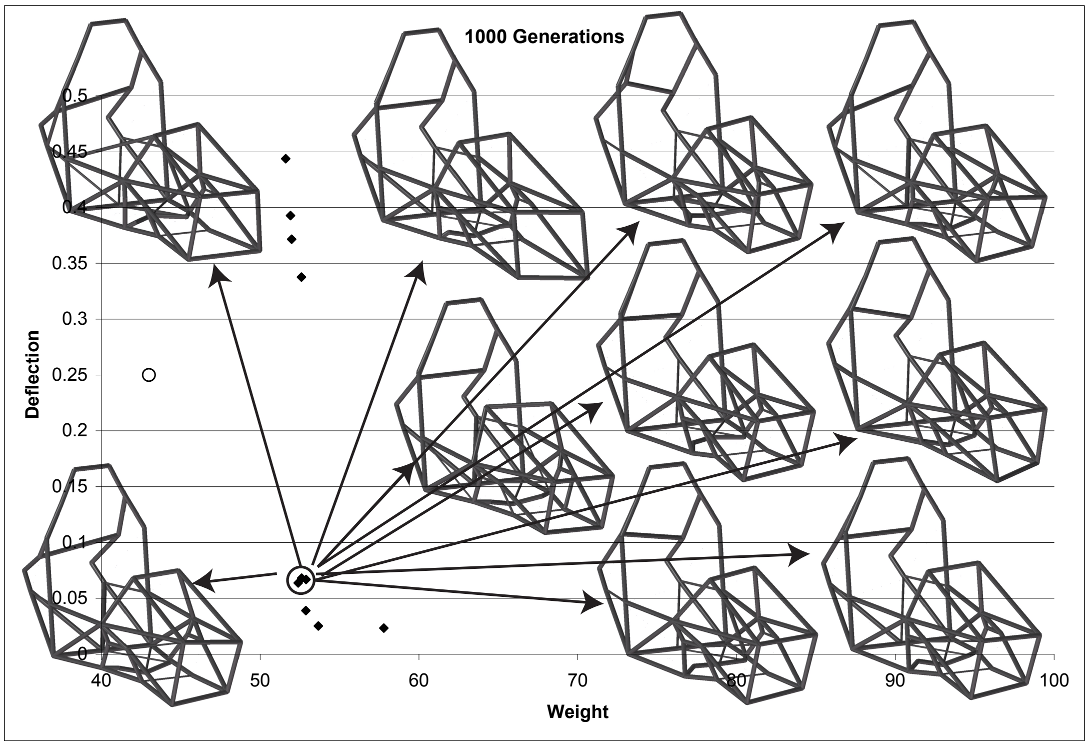


Figure 4.15: Ten iso-performing designs expanded about a single design with performance: 52.89 lbs. weight, 0.0669 in. deflection

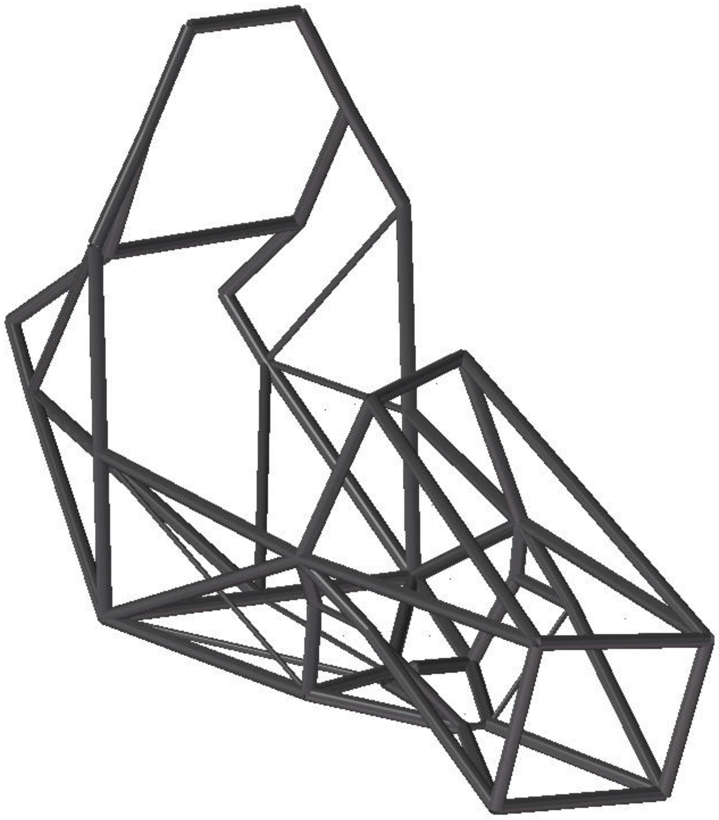


Figure 4.16: Single keel configuration 1 with weight 54.69 lbs. and deflection 0.0296 in.

Chapter 5

Conclusions

Chapter 1 serves as a preview of the chapters that followed. A theoretical framework for coupling parametric CAD and optimization was presented in Chapter 2. In Chapter 3 a practical configuration generator was programmed. This configuration generator could find multiple, iso-performing, feasible electrical contact springs. In Chapter 4, the goals of the two previous papers were combined in a single computer application.

5.1 Introduction Summary

In Chapter 1 we outlined the structure of the this thesis. Chapter 2 is a paper published in the ASME International Mechanical Engineering Conference and Exposition in 2007; in Chapter 2 we developed a design optimization framework that can be used when objectives, constraints, variables, and other conditions are expected to change as the design progresses and new information is gained. The third Chapter is a paper published at the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference in 2006; in Chapter 3 we developed a feasible-configuration generator for multiple-bend, progressive-die-formed springs. Following them in Chapter 4 is the description of a CATIA V5 application that performs optimization constrained updates as described in Chapter 2 and can produce iso-performing designs as described in Chapter 3. Finally is this concluding chapter with an overall summary of conclusions as well as a list of possible future work.

5.2 Handling Frequent Design Changes by Automatic Optimization-Constrained Updates of Parametric CAD Models Summary

The design framework in Chapter 2 is parametric CAD and optimization based. By constraining the parametric updating of CAD models to meet optimization criteria, optimization methods are linked directly to changes that frequently occur during the development of a product. This link is fundamental to the developed framework because it facilitates the repeated optimizations of a part of the development of the system. In essence, if the CAD assembly is changed, the optimization is re-evaluated. One of the main benefits of the proposed method is that it reduces the designer's need to *react* to the design changes in one subsystem by manually correcting the affected design of another. The framework was demonstrated in the development of a suspension rocker for a BYU formula SAE car which was reduced in weight by 18%.

5.3 Feasible Topology Generation For Multiple-Bend Springs Summary

The process of configuration generation can fundamentally affect the success of the design effort, especially if potentially optimal configurations are not at all generated. The purpose of the feasible-configuration generator presented in Chapter 3 is to provide the designer with feasible spring configurations during the spring configuration selection phase of the design process. In this way, the designer may consider a diverse set of feasible configurations before focusing on the detailed design of a select few. By combining principles from concept generation activities, and topology and shape optimization methods, we have developed a process that was able to generate 22 specific spring designs from 11 unique spring classes – all of which feasibly satisfy the functional requirements established for the design of an electronic contact.

5.4 Practical Parametric CAD Based Optimization Framework with Iso-Performing Design Generation Summary

The application in Chapter 4 has been integrated into CATIA V5 in such a way that a CATIA user experienced with line and point creation would be familiar with the geometry creation process. Additional commands necessary to complete a finite element model and optimization setup have also been created. This finite element solver takes advantage of Modified Cholesky's decomposition and removes constrained degrees of freedom in order to reduce runtime. A Genetic Algorithm that uses max-min fitness is used to develop pareto optimal designs and equality constraints are used to develop iso-performing designs. This application was then used to develop a FSAE frame. First, a pareto set of frames was found. Then, one design was selected and iso-performing designs were found. Finally, a parametric change was made to the suspension geometry and a new set of frames with equal performance to that found previously was found.

The end result is a practical framework that can help engineers use what information they have to make the best decisions possible. They can have a geometric model with a coupled analytical model and optimization setup ready in the same time it takes to make just a geometric model. They can then use the optimization routine to explore the design space. They can start broad, finding a pareto frontier of designs, and later narrow in on the performance that they decide is best. They can do all this without worrying about losing their work when someone else makes a change that effects them. They can go to the effort of making a robust parametric model in the beginning. When changes occur, instead of repeating the work they have done before, they can continue adding information to the analytical model and exploring the design space so that they come up with a true global optimum in the end.

5.5 Future Work

The current version of the optimization framework uses the CATIA V5 Visual Basic API, integrated with a C++ genetic algorithm based topology optimization application. In order for this to work, a few intermediate Microsoft Excel [34] and text files must be used. It would improve the speed and robustness of the algorithm if the CATI V5 C++ API were used and directly integrated with the C++ optimization algorithm. In addition, vectors were used instead of arrays throughout the optimization application. In the future, it would be better to use vectors only when dynamic memory allocation is necessary. In a finite element solver, there are many matrices that do not require dynamic memory allocation;

the application would run faster if these were arrays, not vectors. The current Optimization routine begins with a random first generation. The algorithm may converge faster if the first generation were seeded with one design consisting of all members with their lightest cross sectional property, and another design consisting of all members with their stiffest cross sectional property. A similar feature that could improve the utility of the solver would be if it input the latest generation from the last optimization run, and started from that non-random point. A break command that would stop the optimization run and output the results mid-run would also be beneficial. Once the beam element application is more directly integrated into the CATIA V5 C++ API and is solving more efficiently, the next steps would be to upgrade it to handle 2D and 3D elements as well as support displacements, distributed loads, contact, and other different boundary conditions. This would require writing a new solver and replacing the solver component of the framework with the new one.

References

[1] Morino, L., Bernardini, G., and Mastroddi, F., 2006. "Multi-disciplinary optimization for the conceptual design of innovative aircraft configurations." *Computer Modeling in Engineering and Sciences,* **Vol. 13 No. 1**. 1, 9

[2] Hassan, R. A., and Crossley, R. A., 2002. "Multi-objective optimization of conceptual design of communication sattelites with a two-branch tournament genetic algorithm." *43rd AIAA/ASME/ASCE/AHS/ASC Structurs, Structural Dynamics, and Materials Conference,* **2**. 1, 9

[3] Qazi, M., and Linshu, H., 2005. Rapid trajectory optimization using computational intelligence for guidance conceptual design of multistage space launch vehicles Tech. rep., "AIAA Guidance, Navigation and Control Conference". 1, 9

[4] Eschenauer, H., and Olhoff, N., 2001. "Topology optimization of continuum structures: A review." *ASME Appl Mech Rev,* **54**(4), July. 1, 10, 28

[5] Sigmund, O., 2001. "A 99 line topology optimization code written in matlab." *Structural and Multi-Disciplinary Optimization,* **21**. 1, 10

[6] NishiWaki, S., Frecker, M., Min, S., and Kikuchi, N., 1998. "Topology optimization of compliant mechanisms using the homogenization method." *International Journal for Numerical Methods in Engineering,* **42**, pp. 535–559. 1, 10

[7] Bharti, S., and Frecker, M., 2004. "Optimal design and experimental characterization of a compliant mechanism piezoelectric actuator for inertially stabalized rifle." *Journal of Intelligent Material Systems and Structures,* **15**, February. 1, 10

[8] Bowman, K. E., and Mattson, C. A., 2007. Handling frequent design changes by automatic optimization-constrained updates of parametric cad models Tech. Rep. IMECE2007-42379, ASME International Mechanical Engineering Conference and Exposition, November. 3

[9] Mattson, C. A., and Bowman, K. E., 2006. Feasible-configuration generator for multiple-bend springs Tech. Rep. AIAA-2006-7094, 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, September. 4

[10] Mattson, C. A., and Messac, A., 2002. A non-deterministic approach to concept selection using s-Pareto frontiers Tech. rep., ASME 2002 Design Engineering Technical Conference and Computers and Information in Engineering Conference, Design Automation Conference, Montreal, Canada, September. 9

[11] Giunta, A., Eldred, M., Swiler, L., Trucano, T., and Wojtkiewicz, S., 2004. Perspective on optimization under uncertainty: Algorithms and applications Tech. rep., 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, April. 10, 11

[12] Mourelatos, Z., and Jun, Z., 2006. "A design optimization method using evidence theory." *Journal of Mechanical Design,* **128**, July, pp. 901–908. 11

[13] Youn, B., and Wang, P., 2006. Bayesian reliability based design optimization under both aleatory and epistemic uncertainties Tech. Rep. AIAA-2006-6928, 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, September. 11

[14] , 1994-2005. CATIA®Version 5.16 ©Dassault Systémes. All Rights Reserved. 24

[15] , 2008. OptiStruct®Version 8.0 ©Altair Engineering Inc., All Rights Reserved. 24

[16] Mattson, C. A., 2006. Rapid optimization-based conceptualization of multiple-bend spring concepts Tech. Rep. AIAA-2006-2049, AIAA 47th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference, April. 27, 28

[17] Crossley, W. A., and Laananen, D. H., 1996. "Conceptual design of helicopters via genetic algorithm." *Journal of Aircraft,* **33**(6), Nov–Dec, pp. 1060–1070. 27

[18] Mattson, C. A., and Messac, A., 2003. "Concept selection using s-Pareto frontiers." *AIAA Journal,* **41**(6), pp. 1190–1198. 27

[19] Simpson, T. W., Rosen, D., Allen, J. K., and Mistree, F., 1998. "Metrics for assessing design freedom and information certainty in the early stages of design." *ASME Journal of Mechanical Design,* **120**, pp. 628–635. 27

[20] Hassani, B., and Hinton, E., 1999. *Homogenization and Structural Topology Optimization: Theory, Practice, and Software*. Springer-Verlag. 28

[21] Deiter, G. E., 2000. *Engineering Design: A Materials and Processing Approach*. McGraw Hill 3rd Edition. 28

[22] Pahl, G., and Beitz, W., 1996. *Engineering Design: A Systematic Approach.*, second ed. Springer-Verlag, London. 28

[23] Ullman, D. G., 2003. *The Mechanical Design Process*. McGraw-Hill, Inc., New York, pp. 177–207. 28

[24] Rozvany, G. I. N., Bendsoe, M. P., and Kirsch, U., 1995. "Layout optimization of structures." *Applied Mechanics Review,* **48**, pp. 41–119. 28

[25] Osborne, A., 1953. *Applied Imagination*. Charles Scribner and Sons, New York. 29

[26] Zwicky, F., 1948. *The Morphological Method of Analysis and Construction*. Wiley-Interscience, New York Courant Anniversary Volume. 29

[27] Gordon, W. J. J., 1961. *Synectics*. Harper and Row, New York. 29

[28] Firby, P. A., 1991. *Surface Topology*. Ellis Horwood, Chichester. 30

[29] Bhatti, M. A., 2005. *Fundamental finite element analysis and applications: with Mathematica and Matlab computations*. John Wiley and Sons, Hoboken, New Jersey. 32, 33, 54

[30] Messac, A., Melachrinoudis, E., and Sukam, C. P., 2000. "Required relationships between aggregate objective functions and pareto frontiers, with practical implications." *Optimization and Engineering,* **1**(2), June, pp. 171–188. 38

[31] Chen, W., Wiecek, M. M., and Zhang, J., 1999. "Quality utility - a compromise programming approach to robust design." *ASME Journal of Mechanical Design,* **121**, June. 38

[32] Balling, R., 2007. *Computer Structural Optimization*. BYU Academic Publishing. 52, 59, 60, 61

[33] Balling, R., 2007. *Computer Structural Analysis*. BYU Academic Publishing. 58

[34] , 1985-2003. Microsoft®Office Excel 2003 (11.8211.8202) SP3 ©Microsoft Corporation. All Rights Reserved. 77